App.xaml.cs

```
using System.Collections.ObjectModel;
using System.Windows;
using System.Windows.Data;
using Telerik.Windows.Controls;
using Unity;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Unity.Lifetime;
using System.ComponentModel;
using MerlinTestStudio_Demo_Telerik.UserControls.TestConditions;
using MerlinTestStudio_Demo_Telerik.Data.Services;
using MT.TestStudio.GUI.ViewModels;
using UnitConversionLib;
using Telerik.Windows.Input.Touch;
using Telerik.Windows.Automation.Peers;
using System.IO;
using MerlinTestStudio_Demo_Telerik.Data;
using System.Windows.Controls;
using System;
using System.Windows.Input;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;

namespace MerlinTestStudio_Demo_Telerik
{
/// <summary>
/// Interaction logic for App.xaml
/// </summary>
public partial class App : Application
{
public UnityContainer Container { get; set; }
public MainWindowViewModel MWVM { get; set; }

protected override void OnStartup(StartupEventArgs e)
{
//base.OnStartup(e);
#region Unity IoC dependency injection
Container = new UnityContainer();

//Register ViewModels
Container.RegisterType<MainWindowViewModel>();
Container.RegisterType<ConditionsViewModel>();
Container.RegisterType<PinMapViewModel>();
```

```
Container.RegisterType<DUT_TestViewModel>();
Container.RegisterType<SystemConfigViewModel>();
Container.RegisterType<ConnectionDiagramViewModel>();
Container.RegisterType<DataFormattingToolViewModel>();

//Register Services
Container.RegisterType<IViewModelLocator, ViewModelLocator>();
Container.RegisterType<IServiceLocator, ServiceLocator>();
Container.RegisterType<ITestParametersService, TestParametersService>();
Container.RegisterType<ISequenceService, SequenceService>();
Container.RegisterType<IExtensionService, ExtensionService>();
Container.RegisterType<IPinMapService, PinMapService>();
Container.RegisterType<IInstrumentService, InstrumentService>();
Container.RegisterType<IProjectConfigurationService, ProjectConfigurationService>();
Container.RegisterType<IDataFormattingService, DataFormattingService>();
Container.RegisterType<IParameterValueService, ParameterValueService>();
Container.RegisterType<IUnitService, UnitService>();
Container.RegisterType<ITestService, TestService>();
//Container.RegisterType<INotifyPropertyChanged>();

Current.Resources.Add("IoC", Container);
#endregion Unity IoC dependency injection

//Creates all unit conversion definitions to be used everywhere in the application
UnitHelper.CreateUnits();

#region Set Application Settings and preferences
TouchManager.IsTouchEnabled = false; //For GridView speed purposes.
AutomationManager.AutomationMode = AutomationMode.Disabled; //For GridView speed
purposes.

//InitializeComponent();

//Will need to be stored in preferences later.
StyleManager.ApplicationTheme = new VisualStudio2013Theme();
VisualStudio2013Palette.LoadPreset(VisualStudio2013Palette.ColorVariation.Dark);

//VisualStudio2013Palette.Palette.DefaultForegroundColor =
System.Windows.Media.Color.FromRgb(255, 255, 255); //Work around.
#endregion Set Application Settings and preferences

#region Required Directory Checks
```

```
//Deserialize system (systems collection in the future).
if (!Directory.Exists(@"C:\MerlinTest\System Configs\"))
{
Directory.CreateDirectory(@"C:\MerlinTest\System Configs\");
}
//Check the MTS AppData Folder exists before trying to save or load files from it.
if
(!Directory.Exists(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Applicati
onData), "Merlin Test Studio")))
{
Directory.CreateDirectory(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.
ApplicationData), "Merlin Test Studio"));
}
# endregion Required Directory Checks

///Not needed any more...
DataManagement.InitializeDataManger();

#region Window creation and Project load
MainWindow wnd = new MainWindow(); //Create new window manually.
MWVM = (wnd.DataContext as MainWindowViewModel); //Take the data context and store it.
DataStorage.MainViewModel = MWVM;
//Current.Resources.Add("MainVM", MWVM); //set the application resource using the stored data
context.
if (e.Args.Length > 0) //Check to see if user opened app from project file via args.
{
switch (Path.GetExtension(e.Args[0]))
{
case BackendConstants.MerlinProjectExtension:
MWVM.LoadMerlinProject(e.Args[0]);
break;
case ".mtp":
MWVM.LoadLegacyProjectMTP(e.Args[0]);  //Convert project.
break;
default: //Do Nothing
break;
}
}
wnd.Show();
#endregion Window creation and Project load

//KeyBinding keyBinding = new KeyBinding(RadDockingCommands.ClosePane, new
```

```
KeyGesture(Key.F4, ModifierKeys.Control)) { CommandParameter =
ClosePaneMode.DocumentPanes };
//CommandManager.RegisterClassInputBinding(typeof(RadDocking), keyBinding);
}
}
}
```

Converters.cs

```
using System;
using System.Linq;
using System.Text;
using System.Windows.Data;
using System.Windows;
using System.Runtime.InteropServices;
using System.Windows.Interop;
using System.ComponentModel;
using System.Diagnostics;
using System.Windows.Controls.Primitives;
using System.Windows.Media;
using System.Windows.Controls;
using System.Globalization;
using System.Windows.Media.Imaging;

namespace MerlinTestStudio_Demo_Telerik.Converters
{
public class UriToCachedImageConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
if (value == null)
return null;

if (!string.IsNullOrEmpty(value.ToString()))
{
BitmapImage bi = new BitmapImage();
bi.BeginInit();
bi.UriSource = new Uri(value.ToString());
bi.CacheOption = BitmapCacheOption.OnLoad;
bi.EndInit();
return bi;
}
```

```
return null;
}

public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
throw new NotImplementedException("Two way conversion is not supported.");
}
}

/// <summary>
/// Custom converter to convert between an ENUM and boolean and from boolean to an ENUM.
This is used to map the combiner mode enum to a series
/// of radio buttons in the GUI.
/// </summary>
public class EnumBooleanConverter : IValueConverter
{
// Code below based on the information in the following article.
http://stackoverflow.com/questions/397556/how-to-bind-radiobuttons-to-an-enum
// It was modified as I think a bug exists in the two commented out line.

/// <summary>
/// Converts the ENUM parameter to true, false or unsetvalue.
/// </summary>
/// <param name="value"></param>
/// <param name="targetType"></param>
/// <param name="parameter"></param>
/// <param name="culture"></param>
/// <returns></returns>
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
//string parameterString = parameter as string;

string parameterString = parameter.ToString();

if (parameterString == null)
return DependencyProperty.UnsetValue;

if (Enum.IsDefined(value.GetType(), value) == false)
return DependencyProperty.UnsetValue;
```

```
object parameterValue = Enum.Parse(value.GetType(), parameterString);

return parameterValue.Equals(value);
}

/// <summary>
/// Converts boolean to an ENUM based on the fixed ENUM value each radiobutton was assinged.
/// </summary>
/// <param name="value"></param>
/// <param name="targetType"></param>
/// <param name="parameter"></param>
/// <param name="culture"></param>
/// <returns></returns>
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
//string parameterString = parameter as string;

string parameterString = parameter.ToString();

if (parameterString == null || value.Equals(false))
{
return DependencyProperty.UnsetValue;
}

return Enum.Parse(targetType, parameterString);
}
}

/// <summary>
/// Custom converter to convert between an ENUM and boolean and from boolean to an ENUM.
This is used to map the combiner mode enum to a series
/// of radio buttons in the GUI.
/// </summary>
public class InverseEnumBooleanConverter : IValueConverter
{
// Code below based on the information in the following article.
http://stackoverflow.com/questions/397556/how-to-bind-radiobuttons-to-an-enum
// It was modified as I think a bug exists in the two commented out line.

/// <summary>
```

```
/// Converts the ENUM parameter to true, false or unsetvalue.
/// </summary>
/// <param name="value"></param>
/// <param name="targetType"></param>
/// <param name="parameter"></param>
/// <param name="culture"></param>
/// <returns></returns>
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
//string parameterString = parameter as string;

string parameterString = parameter.ToString();

if (parameterString == null)
return DependencyProperty.UnsetValue;

if (Enum.IsDefined(value.GetType(), value) == false)
return DependencyProperty.UnsetValue;

object parameterValue = Enum.Parse(value.GetType(), parameterString);

return !parameterValue.Equals(value);
}

/// <summary>
/// Converts boolean to an ENUM based on the fixed ENUM value each radiobutton was assinged.
/// </summary>
/// <param name="value"></param>
/// <param name="targetType"></param>
/// <param name="parameter"></param>
/// <param name="culture"></param>
/// <returns></returns>
public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
//string parameterString = parameter as string;

string parameterString = parameter.ToString();

if (parameterString == null || value.Equals(false))
{
```

```
return DependencyProperty.UnsetValue;
}

return Enum.Parse(targetType, parameterString);
}
}

/// <summary>
/// Class that defines a single boolean property as a DependencyProperty. This allows simple userControls to be extended ( attached ). This allows a
/// simple way of providing a templated style to multiple controls when the name of the binding property is different. In the style we bind to this
/// property and in the XAML we bind the property to this property.
/// </summary>
public class Attached
{
public static readonly DependencyProperty IsBubbleSourceProperty = DependencyProperty.RegisterAttached("IsBubbleSource", typeof(Boolean), typeof(Attached), new FrameworkPropertyMetadata(false, FrameworkPropertyMetadataOptions.AffectsRender));

public static void SetIsBubbleSource(UIElement element, Boolean value)
{
element.SetValue(IsBubbleSourceProperty, value);
}
public static Boolean GetIsBubbleSource(UIElement element)
{
return (Boolean)element.GetValue(IsBubbleSourceProperty);
}
}

public class ContentToMarginConverter : IValueConverter
{
#region IValueConverter Members

public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
{
return new Thickness(0, 0, -((ContentPresenter)value).ActualHeight, 0);
}

public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
```

```
{
throw new NotImplementedException();
}

#endregion
}

public class ContentToPathConverter : IValueConverter
{
#region IValueConverter Members

public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
var ps = new PathSegmentCollection(4);
ContentPresenter cp = (ContentPresenter)value;
double h = cp.ActualHeight > 10 ? 1.4 * cp.ActualHeight : 10;
double w = cp.ActualWidth > 10 ? 1.25 * cp.ActualWidth : 10;
ps.Add(new LineSegment(new System.Windows.Point(1, 0.7 * h), true));
ps.Add(new BezierSegment(new System.Windows.Point(1, 0.9 * h), new
System.Windows.Point(0.1 * h, h), new System.Windows.Point(0.3 * h, h), true));
ps.Add(new LineSegment(new System.Windows.Point(w, h), true));
ps.Add(new BezierSegment(new System.Windows.Point(w + 0.6 * h, h), new
System.Windows.Point(w + h, 0), new System.Windows.Point(w + h * 1.3, 0), true));

PathFigure figure = new PathFigure(new System.Windows.Point(1, 0), ps, false);
PathGeometry geometry = new PathGeometry();
geometry.Figures.Add(figure);

return geometry;
}

public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
throw new NotImplementedException();
}

#endregion
}

/// <summary>
```

```
/// Custom converter to return true if two strings are the same and false if they are not.
/// </summary>
public class ModelCodeBooleanConverter : IValueConverter
{
/// <summary>
/// Converts the string parameter to true if the value string is the same, false or unsetvalue
otherwise.
/// </summary>
/// <param name="value"></param>
/// <param name="targetType"></param>
/// <param name="parameter"></param>
/// <param name="culture"></param>
/// <returns></returns>
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
//string parameterString = parameter as string;

//string parameterString = parameter.ToString();
Int16[] paramterArray = parameter as Int16[];
Int16 valueInt = (Int16)value;

if (paramterArray == null)
return DependencyProperty.UnsetValue;

foreach (Int16 myValue in paramterArray)
{
if (myValue == valueInt)
{
return true;
}
}

return false;
}

/// <summary>
/// Helper method so that the boolean converter can accept multiple INPUTS...A | B | C then return
true. Solution proposed :
/// http://stackoverflow.com/questions/18864627/how-to-pass-multiple-converter-parameter-in-
generic-enum-to-boolean-converter
/// </summary>
```

```
/// <param name="enumType"></param>
/// <param name="value"></param>
/// <returns></returns>
private static Enum[] ParseObjectToEnum(Type enumType, object value)
{
var enumValue = value as Enum;
if (enumValue != null)
{
return new[] { enumValue };
}

var str = value as string;
if (string.IsNullOrEmpty(str))
{
throw new ArgumentException("parameter");
}

string[] strArray = str.Split(new[] { ';', ',' },
StringSplitOptions.RemoveEmptyEntries);
var enumArray = new Enum[strArray.Length];
for (int i = 0;
i < strArray.Length;
i++)
{
enumArray[i] = (Enum)Enum.Parse(enumType,
strArray[i],
true);
}

return enumArray;
}

/// <summary>
/// Not in use. As the StringComparator converter compares two strings there is no way to convert
back from a bool to the original string.
/// </summary>
/// <param name="value"></param>
/// <param name="targetType"></param>
/// <param name="parameter"></param>
/// <param name="culture"></param>
/// <returns>Returns null.</returns>
public object ConvertBack(object value, Type targetType, object parameter,
```

```
System.Globalization.CultureInfo culture)
{
return null;
}
}

/// <summary>
/// Custom implementation of BooleanToVisibility that provides a little more control than the MS one. Taken from:
/// http://kentb.blogspot.co.uk/2011/02/booleantovisibilityconverter.html
/// </summary>
public sealed class BooleanToVisibilityConverter : IValueConverter
{
public bool IsReversed { get; set; }
public bool UseHidden { get; set; }
public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
{
var val = System.Convert.ToBoolean(value, CultureInfo.InvariantCulture);
if (this.IsReversed)
{
val = !val;
}
if (val)
{
return Visibility.Visible;
}
return this.UseHidden ? Visibility.Hidden : Visibility.Collapsed;
}
public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
{
throw new NotImplementedException();
}
}
}
```

MainWindow.xaml.cs

```
using System;
using System.Windows;
using Telerik.Windows.Controls;
using MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager;
using MerlinTestStudio_Demo_Telerik.UserControls.TestConditions;
using MerlinTestStudio_Demo_Telerik.UserControls.TestLimits;
```

```
using MerlinTestStudio_Demo_Telerik.UserControls.TestSequences;
using MerlinTestStudio_Demo_Telerik.UserControls;
using System.Windows.Input;
using System.Windows.Controls;
using MT.TestStudio.GUI;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Windows.Media;
using System.Windows.Data;
using System.Windows.Media.Imaging;
using MT.TestStudio.GUI.User_Controls;
using MT.TestStudio.GUI.ViewModels;
using System.IO;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Windows.Navigation;
using Unity;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using Telerik.Windows.Controls.Docking;
using System.Linq;
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;

namespace MerlinTestStudio_Demo_Telerik
{
/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : Window, IView
{
#region Private Member Variables

/// <summary>
/// Instance of this VIEWs VIEW-MODEL;
/// </summary>
//private MainWindowViewModel viewModelObj;

/// <summary>
/// Some images require a context for where they load images from. By setting the app base URI
all images can be loaded.
/// </summary>
//private Uri baseUri;
```

```
#endregion Private Member Variables

//Initialize Component
#region Constructor

public MainWindow()
{
var dataContext = new MainWindowViewModel();
dataContext.View = this as IView;
this.DataContext = dataContext;

//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<MainWindowViewModel>();

//viewModelObj = (DataContext as MainWindowViewModel); //Set viewModelObj for window.
//DataStorage.MainViewModel = (DataContext as MainWindowViewModel); //Set MainViewModel
for entire application reference.

// Set the ViewModels view property to be equal to this. This allows the viewModel access to the
view but only
// through a controled interface thus not breaking the MVVM pattern.
//viewModelObj.View = this as IView;

InitializeComponent();

// Subscribe to the VIEW's loaded event and execute a method that in turn will execute a method
in the View-Model when the View is loaded.
this.Loaded += new RoutedEventHandler(MainWindow_Loaded);

//EventManager.RegisterClassHandler(typeof(Window), Keyboard.KeyUpEvent, new
KeyEventHandler(keyUp), true);

// Set the app's base URI.
//baseUri = BaseUriHelper.GetBaseUri(this);
#if !DEBUG
//Collapse the EZ access dev mode toggle.
DevModeEzAccessToggle.Visibility = Visibility.Collapsed;
#endif
}

#endregion
```

```
//Menu Item Click Events (Currently not in use).
#region Menu Bar Theme Events (Artifacts from 2019)

private void LightTheme_Click(object sender, RoutedEventArgs e)
{
//default color variation
VisualStudio2013Palette.LoadPreset(VisualStudio2013Palette.ColorVariation.Light);

// Set Non-Telerik controls colours.
MainWindowViewModel mwvm = this.DataContext as MainWindowViewModel;

if (mwvm != null)
{
mwvm.NonTelerikControlBackGroundColor = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#EEEEF2"));
}
}

private void DarkTheme_Click(object sender, RoutedEventArgs e)
{
//dark color variation
VisualStudio2013Palette.LoadPreset(VisualStudio2013Palette.ColorVariation.Dark);

// Set Non-Telerik controls colours.
MainWindowViewModel mwvm = this.DataContext as MainWindowViewModel;

if (mwvm != null)
{
mwvm.NonTelerikControlBackGroundColor = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#252526"));
}

}

#endregion

/// <summary>
/// Method that indicates to the view model that the view has been fully loaded.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MainWindow_Loaded(object sender, RoutedEventArgs e)
```

```
{
Style _style = null;

_style = (Style)Resources["CustomWindowStyle"];

this.Style = _style;

var viewModel = this.DataContext as MainWindowViewModel;
if (viewModel != null)
{
viewModel.LoadLayout();
}
}

/// <summary>
/// Closes the current window.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CloseClick(object sender, RoutedEventArgs e)
{
MainWindowViewModel mwvm = this.DataContext as MainWindowViewModel;

if ( mwvm != null)
{
mwvm.ExitProgramCommandExecute();
}

//var window = (Window)((FrameworkElement)sender).TemplatedParent;
//window.Close();
}

/// <summary>
/// Sets the Window size to be normal or maximized.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MaximizeRestoreClick(object sender, RoutedEventArgs e)
{
var window = (Window)((FrameworkElement)sender).TemplatedParent;
if (window.WindowState == System.Windows.WindowState.Normal)
{
```

```
window.WindowState = System.Windows.WindowState.Maximized;
}
else
{
window.WindowState = System.Windows.WindowState.Normal;
}
}

/// <summary>
/// Minimizes the current window.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MinimizeClick(object sender, RoutedEventArgs e)
{
var window = (Window)((FrameworkElement)sender).TemplatedParent;
window.WindowState = System.Windows.WindowState.Minimized;
}


private void mainDocking_PreviewClose(object sender,
Telerik.Windows.Controls.Docking.StateChangeEventArgs e)
{
try
{
//int panesCount = DataStorage.MainViewModel.Panes.Count();
//RadPane rp = e.Panes.FirstOrDefault();
//if (rp != null)
//{
//    PVM pvm = (PVM)rp.DataContext;
//    if (pvm.IsDocument)
//    {
//        pvm.PaneClose();
//        //DataStorage.Panes.Remove(pvm);
//
//        panesCount = DataStorage.MainViewModel.Panes.Count();
//
//        //GC.Collect();
//    }
//}
//else { Console.WriteLine("RadPane on Preview Close is null..."); }
```

```
////////////////////////////////////////////////////////////////////////////////

//// Get a reference to the mainwindowViewModel.
//MainWindowViewModel mwvm = this.DataContext as MainWindowViewModel;
//
//foreach (var pane in e.Panes)
//{
//    UserControl uc = pane.Content as UserControl;
//
//    if (uc != null)
//    {
//        UcViewModelBase viewModel = uc.DataContext as UcViewModelBase;
//
//        if (viewModel != null)
//        {
//            if (viewModel.SaveRequired == true)
//            {
//                string sMessageBoxText = "Save Changes to " +
Path.GetFileName(viewModel.FileName) + "?";
//                string sCaption = "Merlin Test Studio";
//
//                MessageBoxButton btnMessageBox = MessageBoxButton.YesNoCancel;
//                MessageBoxImage icnMessageBox = MessageBoxImage.Warning;
//
//                MessageBoxResult rsltMessageBox = MessageBox.Show(sMessageBoxText,
sCaption, btnMessageBox, icnMessageBox);
//
//                switch (rsltMessageBox)
//                {
//                    case MessageBoxResult.Yes:
//                        {
//                            viewModel.SaveDataFile();
//                            break;
//                        }
//                    case MessageBoxResult.No:
//                        {
//                            // User doesn't want to save changes to indicate a load is required.
//                            viewModel.IsDataLoaded = false;
//                            //viewModel.SaveRequired = false;
//
//                            if (mwvm != null)
//                            {
```

```
//                          mwvm.UpdateProjectTree(viewModel.FileName, false);
//                      }
//                  }
//                  break;
//
//              case MessageBoxResult.Cancel:
//                  {
//                      e.Handled = true;
//                  }
//                  break;
//          }
//      }
//  }
//}
}
catch (Exception ex)
{
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}
}

private void RadPaneMaxMinButton_Click(object sender, RoutedEventArgs e)
{
var btn = sender as Button;
var group = btn.ParentOfType<RadPaneGroup>();
if (group != null)
{
var pane = group.SelectedPane;
if (!pane.IsFloating)
{
MaximizeExtensions.SetLastUsedGroup(pane, group);
pane.MakeFloatingDockable();
WindowCommands.Maximize.Execute(true, pane);

//go maximize
}
}
else
{
//// its toolwindow
var toolWindow = btn.ParentOfType<Telerik.Windows.Controls.Docking.ToolWindow>();
```

```
var splitcontainer = toolWindow.Content as RadSplitContainer;
var paneGroup = splitcontainer.Items[0] as RadPaneGroup;
var pane = paneGroup.SelectedPane;
var lastGroup = MaximizeExtensions.GetLastUsedGroup(pane);
if (lastGroup != null)
{
pane.RemoveFromParent();
lastGroup.Items.Add(pane);
}
else
{
if (toolWindow.WindowState == WindowState.Maximized)
{
WindowCommands.Restore.Execute(true, pane);
}
else
{
WindowCommands.Maximize.Execute(true, pane);
}
}
}
}

#region Docking Events & Methods
private void mainDocking_PreviewShowCompass(object sender,
Telerik.Windows.Controls.Docking.PreviewShowCompassEventArgs e)
{
bool isRootCompass = e.Compass is RootCompass;
var splitContainer = e.DraggedElement as RadSplitContainer;
if (splitContainer != null)
{
bool isDraggingDocument = splitContainer.EnumeratePanes().Any(p => p is RadDocumentPane);
var isTargetDocument = e.TargetGroup == null ? true : e.TargetGroup.EnumeratePanes().Any(p
=> p is RadDocumentPane);
if (isDraggingDocument)
{
e.Canceled = isRootCompass || !isTargetDocument;
}
else
{
e.Canceled = !isRootCompass && isTargetDocument;
}
```

```
}
}

private void mainDocking_Close(object sender,
Telerik.Windows.Controls.Docking.StateChangeEventArgs e)
{
var radPanes = (from p in e.Panes
where p.DataContext is PVM
where (p.DataContext as PVM).IsDocument
select p).ToList(); //Should get the Corresponding RadPanes.
var documentPVMs = e.Panes.Select(p => p.DataContext).OfType<PVM>().Where(vm =>
vm.IsDocument).ToList(); //Should get the corresponding PVMs.
for (int i = 0; i < radPanes.Count(); i++)
{
//Remove MTS Stuff.
documentPVMs[i].PaneClose();
DataStorage.MainViewModel.Panes.Remove(documentPVMs[i]);

//Nullify Telerik GUI stuff. //Already null when removing from collection.
//radPanes[i].Content = null;
//radPanes[i].Header = null;
//radPanes[i].DataContext = null;
//radPanes[i].RemoveFromParent();

///STILL NOT GC'ing correctly.
//Forcing a GC does not work.
//GC.Collect();
}
//foreach (PVM pvm in documentPVMs)
//{
//    pvm.PaneClose();
//    DataStorage.Panes.Remove(pvm);
//}
}

private void FilterActiveViewsSource(object sender, System.Windows.Data.FilterEventArgs e)
{
var vm = e.Item as PVM;
e.Accepted = vm.IsDocument;
}

private void FilterToolboxesSource(object sender, System.Windows.Data.FilterEventArgs e)
```

```
{
var vm = e.Item as PVM;
e.Accepted = !vm.IsDocument;
}

public RadDocking GetRadDocking()
{
return this.mainDocking;
}

#endregion

//Application Main Window Closing.
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
MainWindowViewModel mwvm = this.DataContext as MainWindowViewModel;

//Save the system configuration on close.
mwvm.MySystem.SaveToXml(BackendConstants.SystemFilePath);
mwvm.MyRecentData.SaveToXml(BackendConstants.RecentsDataFilePath);
}
}
}
```

RelayCommands.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
using System.Windows.Input;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
//  Original author - Josh Smith - http://msdn.microsoft.com/en-
us/magazine/dd419663.aspx#id0090030

/// <summary>
/// A command whose sole purpose is to relay its functionality to other objects by invoking
delegates. The default return value for the CanExecute method is 'true'.
/// </summary>
```

```
public class RelayCommand<T> : ICommand
{

#region Declarations

readonly Predicate<T> _canExecute;
readonly Action<T> _execute;

#endregion

#region Constructors

/// <summary>
/// Initializes a new instance of the <see cref="RelayCommand&lt;T&gt;"/> class and the command can always be executed.
/// </summary>
/// <param name="execute">The execution logic.</param>
public RelayCommand(Action<T> execute)
: this(execute, null)
{
}

/// <summary>
/// Initializes a new instance of the <see cref="RelayCommand&lt;T&gt;"/> class.
/// </summary>
/// <param name="execute">The execution logic.</param>
/// <param name="canExecute">The execution status logic.</param>
public RelayCommand(Action<T> execute, Predicate<T> canExecute)
{

if (execute == null)
throw new ArgumentNullException("execute");
_execute = execute;
_canExecute = canExecute;
}

#endregion

#region ICommand Members

public event EventHandler CanExecuteChanged
{
```

```
add
{

if (_canExecute != null)
CommandManager.RequerySuggested += value;
}
remove
{

if (_canExecute != null)
CommandManager.RequerySuggested -= value;
}
}

[DebuggerStepThrough]
public Boolean CanExecute(Object parameter)
{
return _canExecute == null ? true : _canExecute((T)parameter);
}

public void Execute(Object parameter)
{
_execute((T)parameter);
}

#endregion
}

/// <summary>
/// A command whose sole purpose is to relay its functionality to other objects by invoking
delegates. The default return value for the CanExecute method is 'true'.
/// </summary>
public class RelayCommand : ICommand
{

#region Declarations

readonly Func<Boolean> _canExecute;
readonly Action _execute;

#endregion
```

```
#region Constructors

/// <summary>
/// Initializes a new instance of the <see cref="RelayCommand&lt;T&gt;"/> class and the command can always be executed.
/// </summary>
/// <param name="execute">The execution logic.</param>
public RelayCommand(Action execute)
: this(execute, null)
{
}

/// <summary>
/// Initializes a new instance of the <see cref="RelayCommand&lt;T&gt;"/> class.
/// </summary>
/// <param name="execute">The execution logic.</param>
/// <param name="canExecute">The execution status logic.</param>
public RelayCommand(Action execute, Func<Boolean> canExecute)
{

if (execute == null)
throw new ArgumentNullException("execute");
_execute = execute;
_canExecute = canExecute;
}

#endregion

#region ICommand Members

public event EventHandler CanExecuteChanged
{
add
{

if (_canExecute != null)
CommandManager.RequerySuggested += value;
}
remove
{

if (_canExecute != null)
```

```
CommandManager.RequerySuggested -= value;
}
}

[DebuggerStepThrough]
public Boolean CanExecute(Object parameter)
{
return _canExecute == null ? true : _canExecute();
}

public void Execute(Object parameter)
{
_execute();
}

#endregion
}
}
```

ServiceLocator.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Services;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Unity;

namespace MerlinTestStudio_Demo_Telerik
{
public interface IServiceLocator
{

}

public class ServiceLocator : IServiceLocator
{
private UnityContainer _container;

public ServiceLocator()
{
```

```
_container = (UnityContainer)Application.Current.Resources["IoC"];
}

public ISequenceService sequenceService { get { return
_container.Resolve<ISequenceService>(); } }

public ITestParametersService testParameterService { get { return
_container.Resolve<ITestParametersService>(); } }

}
}
```

ViewModelLocator.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Windows;
using Unity;

namespace MerlinTestStudio_Demo_Telerik
{
public interface IViewModelLocator
{
/// <summary>
/// Gets the viewmodel instance from the datacontext of the passed
/// usercontrol name by searching the ProjectTree.
/// </summary>
/// <param name="UsercontrolName"></param>
object GetUsercontrolDataContext(string UsercontrolName);

void OnControlInteract(string UsercontrolName);

string GetProjectFilePath();
}

/// <summary>
/// Class to locate view models throughout the application.
/// </summary>
public class ViewModelLocator : IViewModelLocator
{
//Temporary instance getter.
public MainWindowViewModel _mwvm { get; private set; }
private UnityContainer _container;
```

```
//Constructor
public ViewModelLocator()
{
_mwvm = (MainWindowViewModel)App.Current.MainWindow.DataContext;
_container = (UnityContainer)Application.Current.Resources["IoC"];
}

/// <summary>
/// Gets the viewmodel instance from the datacontext of the passed
/// usercontrol name by searching the ProjectTree.
/// </summary>
/// <param name="UsercontrolName"></param>
public object GetUsercontrolDataContext(string UsercontrolName)
{
//var projTree = _mwvm.ProjectTreeNodes.ToList()[0].Children;
//
//foreach (ProjectTree pt_under in projTree.SelectMany(pt => pt.Children.Where(pt_under =>
pt_under.Name == UsercontrolName).Select(pt_under => pt_under)))
//{
//    return pt_under.UcControl.DataContext;
//}

return null;
}

public void OnControlInteract(string UsercontrolName)
{
//var projTree = _mwvm.ProjectTreeNodes.ToList()[0].Children;
//foreach (ProjectTree pt_under in projTree.SelectMany(pt => pt.Children.Where(pt_under =>
pt_under.Name == UsercontrolName).Select(pt_under => pt_under)))
//{
//    pt_under.SaveRequired = true;
//}
}



public string GetProjectFilePath()
{
//string projectFileName = _mwvm.CurrentProjectFile;
//string projectFilePath = _mwvm.CurrentProjectDirectory;
//return (projectFilePath + Path.GetFileNameWithoutExtension(projectFileName));
return null;
```

```
}

public MainWindowViewModel MWVM { get { return _mwvm; } }
public DataFormattingToolViewModel DFT_VM { get { return
_container.Resolve<DataFormattingToolViewModel>(); } }
public PinMapViewModel PinMap_VM { get { return _container.Resolve<PinMapViewModel>(); } }
}
}
```

BackendConstants.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.UserControls.PaneViews;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using Telerik.Windows.Controls.Docking;

namespace MerlinTestStudio_Demo_Telerik.Data
{
public enum ProjectFileSection
{
Product_Configurations,
Project_Data, //Physical folder currently houses backend cal data for Cal.Defs. (unofficially)
Test_Limits,
Test_Parameters,
Test_Sequences,
Calibration_Definitions,
Calibration_Limits,
Connection_Manager,
Connection_Diagrams,
Waveforms,
Digital_Patterns,
Dll,
```

```
WaveformSubFolder, //Temp. needs to be fully dynamic!
None
}

public enum MerlinSystemType
{
APS_500,
APS_300,
APS_100,

None
}

[Serializable]
[TypeConverter(typeof(EnumDescriptionTypeConverter))]
public enum RffePatternSpec
{
/// <summary>
/// Regular Write.
/// </summary>
[Description("Write")]
W,
/// <summary>
/// Regular Read.
/// </summary>
[Description("Read")]
R,
/// <summary>
/// Zero Write.
/// </summary>
[Description("Zero Write")]
ZW,
/// <summary>
/// Masked Write.
/// </summary>
[Description("Masked Write")]
MW,
/// <summary>
/// Extended Write.
/// </summary>
[Description("Extended Write")]
EW,
```

```
/// <summary>
/// Extended Read.
/// </summary>
[Description("Extended Read")]
ER,
/// <summary>
/// Long Extended Write.
/// </summary>
[Description("Long Exteneded Write")] //Extended Write Long
LEW,
/// <summary>
/// Long Extended Read.
/// </summary>
[Description("Long Extended Read")] //Extended Read Long
LER,
///// <summary>
///// Universal Extended Write.
///// </summary>
//[Description("Universal Extended Write")]
//UEW,
///// <summary>
///// Universal Extended Read.
///// </summary>
//[Description("Universal Extended Read")]
//UER,

/// <summary>
/// If manual is used do not display in the GUI, allow the user to specify manually what the RFFE Pattern Spec should be via combo box.
/// </summary>
//Manual
}

public static class BackendConstants
{
public const double CurrentMerlinProjectVersion = 7; //Change every time MerlinProject class undergoes a critical update or change that effects backwards compatability.

#region File Extension Constants
public const string MerlinProduct_Config_Extension = ".prodconfig";

public const string Test_Limits_Extension = ".limits";
```

```
public const string Gold_Limits_Extension = ".golds";
public const string Offset_Limits_Extension = ".offsets";
public const string Traceloss_Limits_Extension = ".traceloss";
public const string Test_Fixture_Limits_Extension = ".fixtures";

//public const string Test_Parameters_Extension = ".xml";
//public const string Dut_Test_Sequence_Extension = ".dutseq";

//MTCD => Merlin Test Calibration Definition.
public const string MTS_Cal_Config_Extension = ".caldef"; //Requested this be converted from
json to xml format. //.mtcc

public const string Cal_Limit_Extension = ".callimit";

public const string SEDPin32_PinMap_Extension = ".pinmap";
public const string TestStudioPinMapExtension = ".xml";

//public const string MT_Waveform_Extension = ".mtwf"; //No longer used, any .mtwf files loaded
in will be converted to .mtwf2
public const string MT_WaveformV2_Extension = ".mtwf2";
public const string AIQ_Waveform_Extension = ".aiq";

public const string RFFE_Pattern_Extension = ".rffepat";
public const string Digital_Levels_Extension = ".digilevels";
public const string Digital_Timing_Extension = ".digitiming";
#endregion

public const string MerlinProjectExtension = ".mtproj";
//public const string TestStudioCalDefModelExtension = ".mtxml";

public const string InitialCalResultsDirectory = @"C:\MerlinTest\Production\UserCal\";
public const string InitialProjectDirectory = @"C:\MerlinTest\Project";
public const string InitialWaveformDirectory = @"C:\MerlinTest\System\Waveforms\";

//public const string DockingLayoutFileName = "Layout.xml";
public const string DockingDocumentsFileName = "Documents.xml";//Decomissioned.

//Uses application data folder. Perhaps extend the AppData folder path to it's own static variable
and use it for a "User Preferences" XML file.
//public static string RecentsDataFilePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
"RecentsData.xml");
```

```
//public static string SystemFilePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
"System.xml");
public static string RecentsDataFilePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "Merlin
Test Studio", "RecentsData.xml");
public static string UserPreferencesDataFilePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "Merlin
Test Studio", "UserPreferences.xml");
public static string DockingLayoutFilePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "Merlin
Test Studio", "Layout.xml");
public static string SystemFilePath = Path.Combine(@"C:\MerlinTest\System Configs\", "MTS
System Data.xml");

public static List<string> TargetSystemOptions = new List<string>() { "APS-500", "APS-300",
"APS-100" };
public static List<KeyValuePair<MerlinSystemType, string>> TargetSystemOptionsKvPs = new
List<KeyValuePair<MerlinSystemType, string>>()
{
new KeyValuePair<MerlinSystemType, string>(MerlinSystemType.APS_500, "APS-500"),
new KeyValuePair<MerlinSystemType, string>(MerlinSystemType.APS_300, "APS-300"),
new KeyValuePair<MerlinSystemType, string>(MerlinSystemType.APS_100, "APS-100")
};

public static List<string> ExistingInstruments = new List<string>() { "DPS", "PSM", "HPA100",
"RF100", "RF110", "RF200", "RF210", "RF300", "OpenATE", "PE32H", "SEDPin32" };

public static string ProductConfigurationsDirString = "Product Configurations";
public static string ProjectDataDirString = "Project Data"; //Will need for backend purposes.
public static string TestLimitsDirString = "Test Limits";
public static string TestParametersDirString = "Test Parameters";
public static string TestSequencesDirString = "Test Sequence";
public static string CalibrationDefinitionsDirString = "Calibration Definitions";
public static string CalibrationLimitsDirString = "Calibration Limits";
public static string ConnectionManagerDirString = "Pin Map";
public static string ConnectionDiagramsDirString = "Connection Diagrams";
public static string WaveformsDirString = "Waveforms";
public static string DigitalPatternsDirString = "Digital Patterns";
public static string DllDirString = "Dll";
public static List<KeyValuePair<ProjectFileSection, string>> ProjectFileSectionKvPs = new
List<KeyValuePair<ProjectFileSection, string>>()
```

```
{
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Product_Configurations,
ProductConfigurationsDirString),
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Project_Data,
ProjectDataDirString), //Will need for backend purposes.
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Test_Limits,
TestLimitsDirString),
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Test_Parameters,
TestParametersDirString),
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Test_Sequences,
TestSequencesDirString),
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Calibration_Definitions,
CalibrationDefinitionsDirString),
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Calibration_Limits,
CalibrationLimitsDirString),
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Connection_Manager,
ConnectionManagerDirString),
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Connection_Diagrams,
ConnectionDiagramsDirString),
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Waveforms,
WaveformsDirString),
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Digital_Patterns,
DigitalPatternsDirString),
new KeyValuePair<ProjectFileSection, string>(ProjectFileSection.Dll, DllDirString)
};

public static List<string> WaveformSections = new List<string>() { "GSM/EDGE", "C2K",
"1xEvDo", "TD-SCDMA", "UMTS", "LTE-FDD", "LTE-TDD", "NR", "WLAN", "CW" };

public static List<string> CalLimit_Producible_Results = new List<string>()
{
"noiseCalibrationResultsRF110RF210Lf",
"measRf210HfPortAmpConfigurationsResults",
"measRf210LfPortAmpConfigurationsResults",
"measRf110Rf210HfPortAmpConfigurationsResults",
"measRf110Rf210LfPortAmpConfigurationsResults",
"noiseCalibrationResultsRF110HfRF210Lf",
"srcRf110HfPortAmpConfigurationsResults",
"srcRf110PortAmpConfigurationsResults"
};

public static List<string> RFFE_Opcode_Options = new List<string>()
```

```
{
"SET_FLAGS(CPU_A0)",
"SET_FLAGS(CPU_A1)",
"SET_FLAGS(CPU_B0)",
"SET_FLAGS(CPU_B1)",
"SET_FLAGS(CPU_C0)",
"SET_FLAGS(CPU_C1)",
"SET_FLAGS(CPU_D0)",
"SET_FLAGS(CPU_D1)"
};

//The higher number key characters for the identifiers are tested first.
public static List<KeyValuePair<RffePatternSpec, string>> RffePatternSpecKvPs = new
List<KeyValuePair<RffePatternSpec, string>>()
{
new KeyValuePair<RffePatternSpec, string>(RffePatternSpec.LEW, "LEW"),
new KeyValuePair<RffePatternSpec, string>(RffePatternSpec.LER, "LER"),
//new KeyValuePair<RffePatternSpec, string>(RffePatternSpec.UEW, "UEW"),
//new KeyValuePair<RffePatternSpec, string>(RffePatternSpec.UER, "UER"),
new KeyValuePair<RffePatternSpec, string>(RffePatternSpec.ZW, "ZW"),
new KeyValuePair<RffePatternSpec, string>(RffePatternSpec.MW, "MW"),
new KeyValuePair<RffePatternSpec, string>(RffePatternSpec.EW, "EW"),
new KeyValuePair<RffePatternSpec, string>(RffePatternSpec.ER, "ER"),
new KeyValuePair<RffePatternSpec, string>(RffePatternSpec.W, "W"),
new KeyValuePair<RffePatternSpec, string>(RffePatternSpec.R, "R"),
};

#region Instrument Ports
private static ObservableCollection<string> BlankDefaultPorts = new
ObservableCollection<string>() { "Select Instrument" };
private static ObservableCollection<string> Rf100Ports = new
ObservableCollection<string>(){"P1","P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9", "P10",
"P11", "P12", "P13", "P14","P15", "P16", "MA","MB","MC","MD"};
private static ObservableCollection<string> Rf110Ports = new
ObservableCollection<string>(){"P1","P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9", "P10",
"P11", "P12", "P13", "P14","P15", "P16", "HF_P1", "HF_P2", "HF_P3", "HF_P4"};
private static ObservableCollection<string> Rf200Ports = new ObservableCollection<string>(){
"M1","M2", "M3", "M4", "M5", "M6", "M7", "M8", "M9", "M10",
"M11", "M12", "M13", "M14", "M15", "M16", "M17","M18","M19","M20", "M21", "M22", "M23",
"M24", "M25", "M26" };
private static ObservableCollection<string> Rf210Ports = new ObservableCollection<string>(){
"M1","M2", "M3", "M4", "M5", "M6", "M7", "M8", "M9", "M10",
```

```
"M11", "M12", "M13", "M14", "M15", "M16", "HF_M1", "HF_M2", "HF_M3", "HF_M4", "HF_M5",
"HF_M6", "HF_M7", "HF_M8" };
private static ObservableCollection<string> DpsPorts = new ObservableCollection<string>(){
"LPS-1", "LPS-2", "LPS-3", "LPS-4", "LPS-5", "LPS-6", "LPS-7", "LPS-8",
"HPC-1", "HPC-2", "HPC-3", "HPC-4",
"GPIO1","GPIO2","GPIO3","GPIO4","GPIO5","GPIO6","GPIO7","GPIO8","GPIO9","GPIO10","GPI
O11","GPIO12","GPIO13","GPIO14",
"GPIO15","GPIO16","GPIO17","GPIO18","GPIO19","GPIO20","GPIO21","GPIO22","GPIO23","GPI
O24","GPIO25","GPIO26","GPIO27",
"GPIO28","GPIO29","GPIO30","GPIO31","GPIO32",
"AI_14","AI_Sense","12V_Sw_Out","AI_4","AI_8","AI_24","AI_29","AI_31","PFI_1","+5V_Sw_Out",
"28VDC"};
private static ObservableCollection<string> OpenATE_Ports = new
ObservableCollection<string>() //Channels 1 - 32
{
"1", "2", "3", "4", "5", "6","7", "8","9",
"10","11","12","13","14","15","16","17","18","19","20","21","22","23","24","25","26","27","28","29","30
","31","32"
};

#endregion


#region Methods
public static ObservableCollection<string> GetPorts(string instrumentName)
{
if (string.IsNullOrEmpty(instrumentName))
{
Console.WriteLine("Instrument name is null");
return BlankDefaultPorts;
}
string[] caseName = Regex.Split(instrumentName, " "); //Cannot handle null case.
switch (caseName[0])
{
case "RF100":
return Rf100Ports;
case "RF110":
return Rf110Ports;
case "RF200":
return Rf200Ports;
case "RF210":
return Rf210Ports;
```

```
case "OpenATE":
case "PE32H":
case "SEDPin32":
return OpenATE_Ports;
case "DPS":
case "PSM":
return DpsPorts;
default:
return BlankDefaultPorts;
}
}

//Gets the description string value of an enum using the [Description("...")] decorator.
public static string GetDescription<T>(this T enumerationValue)
where T : struct
{
Type type = enumerationValue.GetType();
if (!type.IsEnum)
{
throw new ArgumentException("EnumerationValue must be of Enum type", "enumerationValue");
}

//Tries to find a DescriptionAttribute for a potential friendly name
//for the enum
MemberInfo[] memberInfo = type.GetMember(enumerationValue.ToString());
if (memberInfo != null && memberInfo.Length > 0)
{
object[] attrs = memberInfo[0].GetCustomAttributes(typeof(DescriptionAttribute), false);

if (attrs != null && attrs.Length > 0)
{
//Pull out the description value
return ((DescriptionAttribute)attrs[0]).Description;
}
}
//If we have no description attribute, just return the ToString of the enum
return enumerationValue.ToString();
}
#endregion
}

public static class RandomData
```

```
{
private static readonly Random _random = new Random();

public static void FillWithRandomData<T>(T obj) where T : class
{
var properties = typeof(T).GetProperties();

foreach (var property in properties)
{
if (property.PropertyType == typeof(int))
{
property.SetValue(obj, _random.Next(int.MinValue, int.MaxValue));
}
else if (property.PropertyType == typeof(string))
{
property.SetValue(obj, Guid.NewGuid().ToString());
}
else if (property.PropertyType == typeof(DateTime))
{
property.SetValue(obj, DateTime.Now.AddDays(_random.Next(365)));
}
else if (property.PropertyType == typeof(bool))
{
property.SetValue(obj, _random.Next(2) == 0);
}
else if (property.PropertyType == typeof(double))
{
property.SetValue(obj, _random.NextDouble());
}
else if (property.PropertyType.IsEnum)
{
Array enumValues = Enum.GetValues(property.PropertyType);
property.SetValue(obj, enumValues.GetValue(_random.Next(enumValues.Length)));
}
else
{
// Add additional checks for other property types
}
}
}
}
```

```
}
```

DataManagement.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using Telerik.Windows.Controls;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Collections.ObjectModel;
using System.Xml.Serialization;
using System.Windows;
using System.Linq;
using MerlinTestStudio_Demo_Telerik.Data;
using System.Security.Cryptography;
using System.Windows.Navigation;

namespace MerlinTestStudio_Demo_Telerik
{
public class DataManagement : ViewModelBase
{
private static DataSet dataSet = new DataSet("Project Data Set");

#region Public Methods

/// <summary>
/// Checks to see if all groups have the same attributes.
/// </summary>
/// <param name="bufferDataTable"></param>
public static void GroupAttributeCheck()
{

//var dissolvedGroups = new List<string>();
//List<string> groupList = new List<string>();
//
//foreach (DataRow conRow in bufferDataTable.Rows)
//    if(conRow[4].ToString() !="" && groupList.Contains(conRow[4].ToString()) == false)
//        groupList.Add(conRow[4].ToString());
//
//foreach (string item in groupList)
//{
//    DataRow rowCheck = null;
```

```
//    bool markforDelete = false;
//
//    foreach (DataRow conRow in bufferDataTable.Rows)
//        if (conRow[4].ToString() == item)
//            rowCheck = conRow;
//
//    foreach (DataRow conRow in bufferDataTable.Rows)
//        if (conRow[4].ToString() == item)
//            for (int i = 5; i < bufferDataTable.Columns.Count; i++)
//                if (conRow[i].ToString() != rowCheck[i].ToString())
//                    markforDelete = true;
//
//    if (markforDelete == true)
//    {
//        foreach (DataRow r in bufferDataTable.Rows) //Removes the group name from all existing
tests.
//            if (item == r[4].ToString())
//                r[4] = string.Empty;
//
//        dissolvedGroups.Add(item);
//        markforDelete = false;
//    }
//}
//
//if (dissolvedGroups != null)
//    foreach (var item in dissolvedGroups)
//        System.Windows.MessageBox.Show(item.ToString() + " has been dissolved due to a
condition mismatch.");

}

public static void InitializeDataManger()
{
DataStorage.CreateDefaultColumns();
//dataSet.Tables.Add(DataStorage.m_CSV_Conditions);
dataSet.Tables.Add(DataStorage.m_CSV_Limits);
dataSet.Tables.Add(DataStorage.ProjectInfo);
BuildProjectInfo();
}

//Recode.
public static void BuildProjectInfo()
```

```
{
DataStorage.ProjectInfo.Clear();
string[] conditionsHeader = new string[] { "New Product", "A", DateTime.Now.ToString(), "", "1" };
string[] limitsHeader = new string[] { "New Product", "A", DateTime.Now.ToString(), "", "1", "", "", "",
"", "", "" };
DataRow Row;
Row = DataStorage.ProjectInfo.NewRow();
for (int i = 0; i < 5; i++)
{
Row[i] = conditionsHeader[i];
}
DataStorage.ProjectInfo.Rows.Add(Row);

Row = DataStorage.ProjectInfo.NewRow();
for (int i = 0; i < 10; i++)
{
Row[i] = limitsHeader[i];
}
DataStorage.ProjectInfo.Rows.Add(Row);
}



#endregion

}
}
```

DataStorage.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.AppModels;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Collections.ObjectModel;
using System.Data;
using System.Windows;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik
{
public static class DataStorage
{
```

```
public static MainWindowViewModel MainViewModel { get; set; }//=>
((MainWindowViewModel)Application.Current.Resources["MainVM"]);

//public static Theme CurrentTheme { get; set; }

public static ObservableCollection<Error> ErrorLogs = new ObservableCollection<Error>();

public static MerlinSystem CurrentSystem = new MerlinSystem(); //Is deserialized on
MainWindowViewModel constructor.

public static UserPreferencesData CurrentUserPreferences = new UserPreferencesData();

//public static ObservableCollection<PVM> Panes = new ObservableCollection<PVM>();

//public static ObservableCollection<MerlinProject> Projects = new
ObservableCollection<MerlinProject>();

public static ObservableCollection<CalDataModel> CalDataResults = new
ObservableCollection<CalDataModel>();


//public static ObservableCollection<ISequenceItem> _allGroups = new
ObservableCollection<ISequenceItem>();
//public static ObservableCollection<ITest> _allTests = new ObservableCollection<ITest>();
//public static ObservableCollection<Data.Instrument> _allInstruments = new
ObservableCollection<Data.Instrument>();
//public static ObservableCollection<Data.Pin> _allPins = new ObservableCollection<Data.Pin>();
//public static ObservableCollection<Data.DLL> _allDLLs = new
ObservableCollection<Data.DLL>();
//public static ObservableCollection<Data.ParameterMapTag> _allParameterMapTags = new
ObservableCollection<Data.ParameterMapTag>();

//public static ObservableCollection<Data.TestToolBusinessObject>
_testToolObjectsListSourceTemp = new ObservableCollection<Data.TestToolBusinessObject>();

/// <summary>
/// BinaryFormatter below does not care about the culture or version numbers of the classes and
objects it is deserializing. (stored incase future need)
/// </summary>
//BinaryFormatter bf = new BinaryFormatter() { AssemblyFormat =
System.Runtime.Serialization.Formatters.FormatterAssemblyStyle.Simple };
```

```
public static DataTable m_CSV_Limits = new DataTable("Limits");
//public static DataTable m_CSV_Conditions = new DataTable("Conditions");
public static DataTable ProjectInfo = new DataTable("Project Information");

/// <summary>
/// Creates the default columns in the DataTables.
/// </summary>
public static void CreateDefaultColumns()
{
m_CSV_Limits.Columns.Add("Test Number", typeof(int));
//m_CSV_Limits.PrimaryKey = new DataColumn[] { m_CSV_Limits.Columns["Test Number"] };
m_CSV_Limits.Columns.Add("Test Name", typeof(string));
m_CSV_Limits.Columns.Add("Units", typeof(string));
m_CSV_Limits.Columns.Add("HardBinNumber", typeof(string));
m_CSV_Limits.Columns.Add("HardBinName", typeof(string));
m_CSV_Limits.Columns.Add("HardBinPF", typeof(string));
m_CSV_Limits.Columns.Add("SoftBinNumber", typeof(string));
m_CSV_Limits.Columns.Add("SoftBinName", typeof(string));
m_CSV_Limits.Columns.Add("SoftBinPF", typeof(string));
m_CSV_Limits.Columns.Add("FT Lower Limit", typeof(string));
m_CSV_Limits.Columns.Add("FT Upper Limit", typeof(string));
m_CSV_Limits.Columns.Add("QA Lower Limit", typeof(string));
m_CSV_Limits.Columns.Add("QA Upper Limit", typeof(string));
m_CSV_Limits.Columns.Add("Gold Tolerance", typeof(string));
m_CSV_Limits.Columns.Add("Gold Control", typeof(string));
m_CSV_Limits.Columns.Add("MIN", typeof(string));
m_CSV_Limits.Columns.Add("MAX", typeof(string));
m_CSV_Limits.Columns.Add("Apply Offset", typeof(string));
m_CSV_Limits.Columns.Add("Offset1", typeof(string));
m_CSV_Limits.Columns.Add("Offset2", typeof(string));
m_CSV_Limits.Columns.Add("Offset3", typeof(string));
m_CSV_Limits.Columns.Add("Offset4", typeof(string));
m_CSV_Limits.Columns.Add("Offset5", typeof(string));
m_CSV_Limits.Columns.Add("Offset6", typeof(string));
m_CSV_Limits.Columns.Add("Offset7", typeof(string));
m_CSV_Limits.Columns.Add("Offset8", typeof(string));

//m_CSV_Conditions.Columns.Add("Test Number", typeof(int));
////m_CSV_Conditions.PrimaryKey = new DataColumn[] { m_CSV_Conditions.Columns["Test
Number"] };
//m_CSV_Conditions.Columns.Add("Test Name", typeof(string));
//m_CSV_Conditions.Columns.Add("Units", typeof(string));
```

```
//m_CSV_Conditions.Columns.Add("Sequence function", typeof(string));
//m_CSV_Conditions.Columns.Add("Group name", typeof(string));
//m_CSV_Conditions.Columns.Add("RF Source", typeof(string));
//m_CSV_Conditions.Columns.Add("RF Meas", typeof(string));
//m_CSV_Conditions.Columns.Add("Pin", typeof(string));
//m_CSV_Conditions.Columns.Add("Pout", typeof(string));
//m_CSV_Conditions.Columns.Add("Pout Tolerance", typeof(string));
//m_CSV_Conditions.Columns.Add("Frequency", typeof(string));
//m_CSV_Conditions.Columns.Add("Waveform", typeof(string));

for (int i = 1; i < 12; i++)
ProjectInfo.Columns.Add("Data" + i.ToString(), typeof(string));

}

/// <summary>
/// Clears all data in the data store (including DataTable columns).
/// </summary>
public static void ClearAll()
{
//m_CSV_Conditions.Clear();
//m_CSV_Conditions.Columns.Clear();

m_CSV_Limits.Clear();
m_CSV_Limits.Columns.Clear();

ProjectInfo.Clear();
ProjectInfo.Columns.Clear();

//_allTests.Clear();
//_allDLLs.Clear();
//_allGroups.Clear();
//_allInstruments.Clear();
//_allPins.Clear();
//_allParameterMapTags.Clear();

//_testToolObjectsListSourceTemp.Clear();
}
}
}
```

MerlinSystem.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data
{
[Serializable]
[XmlRoot("MerlinSystem")]
public class MerlinSystem : INotifyPropertyChanged
{
#region Private Members
private ObservableCollection<Instrument> _rF_Instruments = new
ObservableCollection<Instrument>();
private ObservableCollection<Instrument> _pXI_Instruments = new
ObservableCollection<Instrument>();
private ObservableCollection<Instrument> _power_Instruments = new
ObservableCollection<Instrument>();
private MerlinSystemType _systemType = MerlinSystemType.None;
#endregion

#region Public Members
public MerlinSystemType SysType//Implement as dynamic or all three variations static.
{
get { return _systemType; }
set { _systemType = value; OnPropertyChanged("SystemType"); }
}
//[XmlAttribute]
public double Version { get; set; }
public byte[] Hash { get; set; }
public ObservableCollection<Instrument> RF_Instruments { get { return _rF_Instruments; } set {
_rF_Instruments = value; OnPropertyChanged("RF_Instruments"); } }
public ObservableCollection<Instrument> PXI_Instruments { get { return _pXI_Instruments; } set {
_pXI_Instruments = value; OnPropertyChanged("PXI_Instruments"); } }
```

```
public ObservableCollection<Instrument> Power_Instruments { get { return _power_Instruments; }
set { _power_Instruments = value; OnPropertyChanged("Power_Instruments"); } }
#endregion

public MerlinSystem()
{
this.Version = 1;
this.SysType = MerlinSystemType.APS_500; //Remove line once dynamic system serialization is
implemented.
}
public MerlinSystem(MerlinSystemType systemType)
{
this.SysType = systemType;
switch (systemType)
{
case MerlinSystemType.APS_500:
this.RF_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "RF110 (1)", InstrumentType = MTSInstrumentType.RF_Instrument });
this.RF_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "RF210 (1)", InstrumentType = MTSInstrumentType.RF_Instrument });
this.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PE32H (1)", InstrumentType = MTSInstrumentType.PXI });
this.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PE32H (2)", InstrumentType = MTSInstrumentType.PXI });
this.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "SEDPin32 (1)", InstrumentType = MTSInstrumentType.PXI });
this.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "SEDPin32 (2)", InstrumentType = MTSInstrumentType.PXI });
this.Power_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PSM (1)", InstrumentType = MTSInstrumentType.PowerSupply });
break;
case MerlinSystemType.APS_300:
this.RF_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "RF110 (1)", InstrumentType = MTSInstrumentType.RF_Instrument });
this.RF_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "RF210 (1)", InstrumentType = MTSInstrumentType.RF_Instrument });
this.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PE32H (1)", InstrumentType = MTSInstrumentType.PXI });
this.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PE32H (2)", InstrumentType = MTSInstrumentType.PXI });
this.Power_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PSM (1)", InstrumentType = MTSInstrumentType.PowerSupply });
```

```
break;
case MerlinSystemType.APS_100:
this.RF_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "RF100 (1)", InstrumentType = MTSInstrumentType.RF_Instrument });
this.RF_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "RF200 (1)", InstrumentType = MTSInstrumentType.RF_Instrument });
this.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PE32H (1)", InstrumentType = MTSInstrumentType.PXI });
this.Power_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PSM (1)", InstrumentType = MTSInstrumentType.PowerSupply });
break;
}
}


#region Methods

public ObservableCollection<Instrument> GetInstruments()
{
ObservableCollection<Instrument> instruments = new ObservableCollection<Instrument>();
foreach(var i in RF_Instruments) { instruments.Add(i); }
foreach (var i in PXI_Instruments) { instruments.Add(i); }
foreach (var i in Power_Instruments) { instruments.Add(i); }
return instruments;
}
public void ClearInstruments()
{
this.RF_Instruments.Clear();
this.PXI_Instruments.Clear();
this.Power_Instruments.Clear();
}

#region Import / Save Methods

//public void SaveToXml(string Filename)
//{
//    XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());
//
//    // Serialize to disk.
//    using (StreamWriter writer = new StreamWriter(Filename))
//    {
//        xmlSerializer.Serialize(writer, this);
```

```
//    }
//}

/// <summary>
/// Imports a PIN Map file in the XML format.
/// </summary>
/// <param name="PinMapXmlFile">The filename and path of the XML based PIN Map
file.</param>
/// <returns>A calibration object.</returns>
public static MerlinSystem ImportXMLNoModifyCheck(string XmlFile)
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(MerlinSystem));

MerlinSystem systemAfterDeSerialize;

// Deserialize XML file.
using (StreamReader sr = new StreamReader(XmlFile))
{
systemAfterDeSerialize = (MerlinSystem)ser.Deserialize(sr);
}

return systemAfterDeSerialize;
}

// Convert an object to a byte array
public static byte[] ObjectToByteArray(object obj)
{
BinaryFormatter bf = new BinaryFormatter();
using (var ms = new MemoryStream())
{
bf.Serialize(ms, obj);
return ms.ToArray();
}
}

public void SaveToXml(string Filename)
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());
//List<object> CalDataV5PropertiesToHash = new List<object>();
```

```
// Make sure the Hash field is NULL before we compute the object as an array of bytes.
this.Hash = null;

// Populate list of objects used to calculate MD5 Hash
//CalDataV5PropertiesToHash = SavePropertiesToBeHashed();

// Treat the object as a series of bytes.
var objectInBytes = ObjectToByteArray(this);

// Compute a hash of the the byte array.
this.Hash = new MD5CryptoServiceProvider().ComputeHash(objectInBytes);

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(Filename))
{
xmlSerializer.Serialize(writer, this);
}
}

public static MerlinSystem ImportXML(string Filename)
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(MerlinSystem));

MerlinSystem dataAfterDeSer;
List<object> CalDataV5PropertiesToHash = new List<object>();

// Deserialize XML file.
using (StreamReader sr = new StreamReader(Filename))
{
dataAfterDeSer = (MerlinSystem)ser.Deserialize(sr);
}

if (dataAfterDeSer.Version >= 1)
{

// Read back the hash value for the file we just read in.
byte[] hashOfFileReadIn = dataAfterDeSer.Hash;

// When we originally create the hash we do it with the Property hash set to NULL. So
// set it back to NULL before we re-compute the hash.
```

```
dataAfterDeSer.Hash = null;

// Populate list of objects used to calculate MD5 Hash
//CalDataV5PropertiesToHash = ImportPropertiesToBeHashed(dataAfterDeSer);

// Convert object to bytes.
var objectInBytes = ObjectToByteArray(dataAfterDeSer);

// Perform a MD5 hash on the byte array.
var objectHash = new MD5CryptoServiceProvider().ComputeHash(objectInBytes);

// Check to see if the new hash matches the old hash. If not raise an exception.
if (objectHash.SequenceEqual(hashOfFileReadIn) != true)
{
MessageBox.Show("Original File Has been altered since it was created!");
}
}

return dataAfterDeSer;
}

#endregion Import / Save Methods
#endregion

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}

[Serializable]
[XmlRoot("SystemsSerializable")]
public class SystemsSerializable
{
public double Version { get; set; }
public byte[] Hash { get; set; }
public ObservableCollection<MerlinSystem> Systems { get; set; }
```

```
public SystemsSerializable() { Version = 1; } //Deserialization Constructor.
public SystemsSerializable(List<KeyValuePair<MerlinSystemType, string>> sysPresets) //Preset
Systems
{
foreach(KeyValuePair<MerlinSystemType, string> sysType in sysPresets)
{
MerlinSystem presetSystem = new MerlinSystem(sysType.Key);
Systems.Add(presetSystem);
}
}

#region Import / Save Methods

//public void SaveToXml(string Filename)
//{
//    XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());
//
//    // Serialize to disk.
//    using (StreamWriter writer = new StreamWriter(Filename))
//    {
//        xmlSerializer.Serialize(writer, this);
//    }
//}

/// <summary>
/// Imports a PIN Map file in the XML format.
/// </summary>
/// <param name="PinMapXmlFile">The filename and path of the XML based PIN Map
file.</param>
/// <returns>A calibration object.</returns>
public static MerlinSystem ImportXMLNoModifyCheck(string XmlFile)
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(MerlinSystem));

MerlinSystem systemAfterDeSerialize;

// Deserialize XML file.
using (StreamReader sr = new StreamReader(XmlFile))
{
```

```
systemAfterDeSerialize = (MerlinSystem)ser.Deserialize(sr);
}

return systemAfterDeSerialize;
}

// Convert an object to a byte array
public static byte[] ObjectToByteArray(object obj)
{
BinaryFormatter bf = new BinaryFormatter();
using (var ms = new MemoryStream())
{
bf.Serialize(ms, obj);
return ms.ToArray();
}
}

public void SaveToXml(string Filename)
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());
//List<object> CalDataV5PropertiesToHash = new List<object>();

// Make sure the Hash field is NULL before we compute the object as an array of bytes.
this.Hash = null;

// Populate list of objects used to calculate MD5 Hash
//CalDataV5PropertiesToHash = SavePropertiesToBeHashed();

// Treat the object as a series of bytes.
var objectInBytes = ObjectToByteArray(this);

// Compute a hash of the the byte array.
this.Hash = new MD5CryptoServiceProvider().ComputeHash(objectInBytes);

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(Filename))
{
xmlSerializer.Serialize(writer, this);
}
}

public static MerlinSystem ImportXML(string Filename)
```

```
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(MerlinSystem));

MerlinSystem dataAfterDeSer;
List<object> CalDataV5PropertiesToHash = new List<object>();

// Deserialize XML file.
using (StreamReader sr = new StreamReader(Filename))
{
dataAfterDeSer = (MerlinSystem)ser.Deserialize(sr);
}

if (dataAfterDeSer.Version >= 1)
{

// Read back the hash value for the file we just read in.
byte[] hashOfFileReadIn = dataAfterDeSer.Hash;

// When we originally create the hash we do it with the Property hash set to NULL. So
// set it back to NULL before we re-compute the hash.
dataAfterDeSer.Hash = null;

// Populate list of objects used to calculate MD5 Hash
//CalDataV5PropertiesToHash = ImportPropertiesToBeHashed(dataAfterDeSer);

// Convert object to bytes.
var objectInBytes = ObjectToByteArray(dataAfterDeSer);

// Perform a MD5 hash on the byte array.
var objectHash = new MD5CryptoServiceProvider().ComputeHash(objectInBytes);

// Check to see if the new hash matches the old hash. If not raise an exception.
if (objectHash.SequenceEqual(hashOfFileReadIn) != true)
{
MessageBox.Show("Original File Has been altered since it was created!");
}
}

return dataAfterDeSer;
}
```

```
#endregion Import / Save Methods
}

}
```

LoseFocusOnEnterTextBox.cs

```
using System;
using System.Windows.Controls;
using System.Windows.Input;
//using System.Windows.Interactivity;

namespace MerlinTestStudio_Demo_Telerik.Data.AttachedBehaviors
{
//public class LoseFocusOnEnterTextBox : Behavior<TextBox>
//{
//    protected override void OnAttached()
//    {
//        if (this.AssociatedObject != null)
//        {
//            base.OnAttached();
//            this.AssociatedObject.KeyDown += AssociatedObject_KeyDown;
//        }
//    }
//
//    protected override void OnDetaching()
//    {
//        if (this.AssociatedObject != null)
//        {
//            this.AssociatedObject.KeyDown -= AssociatedObject_KeyDown;
//            base.OnDetaching();
//        }
//    }
//
//    private void AssociatedObject_KeyDown(object sender, System.Windows.Input.KeyEventArgs
e)
//    {
//        TextBox textBox = sender as TextBox;
//        if (textBox != null)
//        {
//            if (e.Key == Key.Return)
//            {
```

```
//                if (e.Key == Key.Enter)
//                {
//                    textBox.MoveFocus(new TraversalRequest(FocusNavigationDirection.Next));
//                }
//            }
//        }
//    }
//}
}
```

CustomDragDropBehaviorForFFL.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Telerik.Windows.DragDrop.Behaviors;

namespace MerlinTestStudio_Demo_Telerik.Data
{
public class CustomDragDropBehaviorForFFL : ListBoxDragDropBehavior
{
public override void Drop(DragDropState state)
{
if (!state.IsSameControl)
{
foreach (object o in state.DraggedItems)
{
if (!state.DestinationItemsSource.Contains(o))
state.DestinationItemsSource.Add(o);
}
}
base.Drop(state);
}

public override void DragDropCompleted(DragDropState state)
{
foreach (var item in state.DraggedItems)
{
//state.SourceItemsSource.Remove(item);
}
base.DragDropCompleted(state);
```

```
}

public override bool CanDrop(Telerik.Windows.DragDrop.Behaviors.DragDropState state)
{
return state.IsSameControl;
}

protected override bool IsMovingItems(DragDropState state)
{
return state.IsSameControl;
}
}
}
```

CustomDragDropBehaviorForFSL.cs

```
using Telerik.Windows.DragDrop.Behaviors;
using System.Collections.ObjectModel;

namespace MerlinTestStudio_Demo_Telerik.Data
{
public class CustomDragDropBehaviorForFSL : ListBoxDragDropBehavior
{
public override void Drop(DragDropState state)
{
if (!state.IsSameControl)
{
foreach (object o in state.DraggedItems)
{
if (!state.DestinationItemsSource.Contains(o))
{
if((state.DestinationItemsSource as SequenceCollection<Function>).ParentType ==
SequenceItemType.Block)
{
state.DestinationItemsSource.Add((o as Function).GetUniqueCopy());
}
else
{
state.DestinationItemsSource.Add((o as Function).Clone());
}

}
}
```

```
}

if (state.IsSameControl)
{
foreach (object o in state.DraggedItems)
{
if (state.DestinationItemsSource.Contains(o))
state.SourceItemsSource.Remove(o);
state.DestinationItemsSource.Add(o);
}
base.Drop(state);
}

}

public override void DragDropCompleted(DragDropState state)
{
//if (state.IsSameControl)
//{
//    foreach (var item in state.DraggedItems)
//    {
//        state.SourceItemsSource.Remove(item);
//    }
//    base.DragDropCompleted(state);
//}
base.DragDropCompleted(state);
}

public override bool CanDrop(DragDropState state)
{
if (!state.IsSameControl)
return !state.IsSameControl;
else return state.IsSameControl;
}

protected override bool IsMovingItems(DragDropState state)
{
if (!state.IsSameControl)
return !state.IsSameControl;
else return state.IsSameControl;
}
}
```

```
}
```

BooleanToPassFailConverter.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Data;

namespace MerlinTestStudio_Demo_Telerik.Data.Converters
{
/// <summary>
/// Converts boolean values to the following string: True = "Pass" & False = "Fail".
/// </summary>
public class BooleanToPassFailConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
switch (value)
{
case true:
return "Success"; //"Pass"
case false:
return "-"; //"Fail"
default://If Bool is Nullable
return "N/A";
}
}

public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
switch (value)
{
case "Success": //"Pass"
return true;
case "-": //"Fail"
return false;
default: //If Bool is Nullable
return "N/A";
```

```
}
}
}
}
```

BooleanToYesNoConverter.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Data;

namespace MerlinTestStudio_Demo_Telerik.Data.Converters
{
public class BooleanToYesNoConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
switch (value)
{
case true:
return "Yes";
case false:
return "No";
default://If Bool is Nullable
return "N/A";
}
}

public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
switch (value)
{
case "Yes":
return true;
case "No":
return false;
default: //If Bool is Nullable
return "N/A";
```

```
}
}
}

}
```

FrequencyConverter.cs

```
using System;
using System.Globalization;
using UnitConversionLib;
using System.Windows.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;

namespace MerlinTestStudio_Demo_Telerik.Data.Converters
{
//public class FrequencyConverter : IMultiValueConverter
//{
//    public object Convert(object[] values, Type targetType, object parameter, CultureInfo culture)
//    {
//        if (values.Length < 2)
//        {
//            return null;
//        }
//
//        // Get the frequency value and unit
//        double frequency = System.Convert.ToDouble(values[0]);
//        string fromUnit = System.Convert.ToString(values[1]);
//
//        // Get the desired unit to convert to
//        string toUnit = System.Convert.ToString(parameter);
//
//        // Convert the frequency to the desired unit
//        double convertedFrequency = CalPointModel.ValueUnitModifier(frequency, fromUnit,
toUnit);
//
//        // Return the converted frequency
//        return convertedFrequency;
//    }
//
//    public object[] ConvertBack(object value, Type[] targetTypes, object parameter, CultureInfo
culture)
```

```
//    {
//        // Get the converted frequency value
//        double convertedFrequency = System.Convert.ToDouble(value);
//
//        // Get the desired unit to convert from
//        string fromUnit = System.Convert.ToString(parameter);
//
//        // Convert the frequency from the converted unit back to the original unit
//        double originalFrequency = CalPointModel.ValueUnitModifier(convertedFrequency,
fromUnit, "Hz");
//
//        // Return the original frequency value and unit
//        return new object[] { originalFrequency, "Hz" };
//    }
//}

public class FrequencyConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
{
// Get the frequency value
double frequency = (double)value;

// Get the desired unit to convert to
string toUnit = (string)parameter;

//Unit Hertz = Unit.GetRegisteredUnit("Hz");
// Convert the frequency to the desired unit
//double convertedFrequency = UnitConversion.Convert(frequency, Hertz, toUnit);
double convertedFrequency = UnitHelper.ValueUnitModifier(frequency, "Hz", toUnit);

// Return the converted frequency
return convertedFrequency;
}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
{
// Get the frequency value
double frequency = (double)value;

// Get the unit that the frequency is currently in
string fromUnit = (string)parameter;
```

```
// Convert the frequency from the current unit to Hertz
double convertedFrequency = UnitHelper.ValueUnitModifier(frequency, fromUnit, "Hz");
//double convertedFrequency = UnitConversion.Convert(frequency, fromUnit, Unit.Hertz);

// Return the converted frequency
return convertedFrequency;
}
}

}
```

InverseBooleanToPassFailConverter.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Data;

namespace MerlinTestStudio_Demo_Telerik.Data.Converters
{
/// <summary>
/// Converts boolean values to the following inverse string: False = "Pass" & True = "Fail".
/// </summary>
public class InverseBooleanToPassFailConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
switch (value)
{
case true:
return "Compressed"; //"Fail"
case false:
return "Good"; //"Pass"
default://If Bool is Nullable
return "N/A";
}
}

public object ConvertBack(object value, Type targetType, object parameter,
```

```
System.Globalization.CultureInfo culture)
{
switch (value)
{
case "Compressed": //"Fail"
return true;
case "Good": //"Pass"
return false;
default: //If Bool is Nullable
return "N/A";
}
}
}
}
```

NullToColorConverter.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Data;
using System.Windows.Media;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.Data.Converters
{
public class NullToColorConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
{
if (value == null)
{
return Brushes.IndianRed;
}
else
{
//Needs to be dynamic in getting the Default Foreground Color. This did not work with dark
variation of the theme.
//var brush = VisualStudio2013Palette.Palette.DefaultForegroundColor;
```

```
return Brushes.White;
}
}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
{
return Brushes.White;
//throw new NotImplementedException();
}
}

}
```

NullToVisibilityConverter.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Data;

namespace MerlinTestStudio_Demo_Telerik.Data.Converters
{
public class NullToVisibilityConverter : IValueConverter
{
public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
{
return value == null ? Visibility.Visible : Visibility.Collapsed;
}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
{
return null;
//throw new NotImplementedException();
}
}
}
```

AssemblyLoader.cs

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
/// <summary>
/// Helper class to assist with loading of an assembly & any unresolved dependencies.
/// </summary>
public static class AssemblyLoader
{
private static readonly ConcurrentDictionary<string, bool> AssemblyDirectories = new
ConcurrentDictionary<string, bool>();

static AssemblyLoader()
{
AssemblyDirectories[GetExecutingAssemblyDirectory()] = true;
AppDomain.CurrentDomain.AssemblyResolve += ResolveAssembly;
}

public static Assembly LoadWithDependencies(string assemblyPath)
{
AssemblyDirectories[Path.GetDirectoryName(assemblyPath)] = true;
return Assembly.LoadFile(assemblyPath);
}

//private static Assembly ResolveAssembly(object sender, ResolveEventArgs args)
//{
//    // Try to load from the GAC
//    try
//    {
//        var assembly = Assembly.Load(args.Name);
//        if (assembly != null)
//            return assembly;
//    }
//    catch
//    {
//        // Ignore exception and continue to the next step
//    }
```

```
//
//    // Try to load from the AssemblyDirectories
//    string dependentAssemblyName = args.Name.Split(',')[0] + ".dll";
//    List<string> directoriesToScan = AssemblyDirectories.Keys.ToList();
//
//    foreach (string directoryToScan in directoriesToScan)
//    {
//        string dependentAssemblyPath = Path.Combine(directoryToScan,
dependentAssemblyName);
//        if (File.Exists(dependentAssemblyPath))
//            return LoadWithDependencies(dependentAssemblyPath);
//    }
//
//    return null;
//}


private static Assembly ResolveAssembly(object sender, ResolveEventArgs args)
{
string dependentAssemblyName = args.Name.Split(',')[0] + ".dll";
List<string> directoriesToScan = AssemblyDirectories.Keys.ToList();

foreach (string directoryToScan in directoriesToScan)
{
string dependentAssemblyPath = Path.Combine(directoryToScan, dependentAssemblyName);
if (File.Exists(dependentAssemblyPath))
return LoadWithDependencies(dependentAssemblyPath);
}
return null;
}

private static string GetExecutingAssemblyDirectory()
{
string codeBase = Assembly.GetExecutingAssembly().CodeBase;
var uri = new UriBuilder(codeBase);
string path = Uri.UnescapeDataString(uri.Path);
return Path.GetDirectoryName(path);
}
}
}
```

CommandHandler.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public interface IDoCommand
{
void Execute();
void Undo();
}

public class CommandHandler
{

private List<IDoCommand> commandList = new List<IDoCommand>();
private int index;

public void AddCommand(IDoCommand command)
{
if(index < commandList.Count)
{
commandList.RemoveRange(index, commandList.Count - index);
}

commandList.Add(command);
command.Execute();
index++;
}

public void UndoCommand()
{
if (commandList.Count == 0)
return;
if(index > 0)
{
commandList[index - 1].Undo();
index--;
}
}
```

```
public void RedoCommand()
{
if (commandList.Count == 0)
return;
if(index < commandList.Count)
{
index++;
commandList[index - 1].Execute();
}
}
}
}
```

CustomDockingPanesFactory.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Docking;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public class CustomDockingPanesFactory : DockingPanesFactory
{
protected override void AddPane(RadDocking radDocking, RadPane pane)
{
var paneModel = pane.DataContext as PVM;
if (paneModel != null && !(paneModel.IsDocument))
{
RadPaneGroup group = null;
switch (paneModel.InitialPosition)
{
case DockState.DockedRight:
group = radDocking.SplitItems.ToList().FirstOrDefault(i => i.Control.Name == "rightGroup") as
RadPaneGroup;
if (group != null)
```

```
{
group.Items.Add(pane);

}
return;
case DockState.DockedBottom:
group = radDocking.SplitItems.ToList().FirstOrDefault(i => i.Control.Name == "bottomGroup") as
RadPaneGroup;
if (group != null)
{
group.Items.Add(pane);
}
return;
case DockState.DockedLeft:
group = radDocking.SplitItems.ToList().FirstOrDefault(i => i.Control.Name == "leftGroup") as
RadPaneGroup;
if (group != null)
{
group.Items.Add(pane);
}
return;
case DockState.FloatingDockable:
var fdSplitContainer = radDocking.GeneratedItemsFactory.CreateSplitContainer();
group = radDocking.GeneratedItemsFactory.CreatePaneGroup();
fdSplitContainer.Items.Add(group);
group.Items.Add(pane);
radDocking.Items.Add(fdSplitContainer);
pane.MakeFloatingDockable();
return;
case DockState.FloatingOnly:
var foSplitContainer = radDocking.GeneratedItemsFactory.CreateSplitContainer();
group = radDocking.GeneratedItemsFactory.CreatePaneGroup();
foSplitContainer.Items.Add(group);
group.Items.Add(pane);
radDocking.Items.Add(foSplitContainer);
pane.MakeFloatingOnly();
return;
case DockState.DockedTop:
default:
return;
}
}
```

```
base.AddPane(radDocking, pane);
}

protected override RadPane CreatePaneForItem(object item)
{
if (item is PaneViewModel) //Remove/Comment out once PaneViewModel type is obsolete.
{
var viewModel = item as PaneViewModel;
if (viewModel != null)
{
var pane = viewModel.IsDocument ? new RadDocumentPane() : new RadPane();
pane.DataContext = item;
RadDocking.SetSerializationTag(pane, viewModel.Header);
if (viewModel.ContentUserControl != null)
{
pane.Content = viewModel.ContentUserControl;
if ((pane.Content as UserControl).DataContext is IPVMLink)
{
((pane.Content as UserControl).DataContext as IPVMLink).PVM = viewModel;
}
}
else if (viewModel.DataContextType != null && viewModel.ContentType != null) //For waveforms
use case.
{
viewModel.ContentUserControl =
((UserControl)Activator.CreateInstance(viewModel.ContentType, new object[] {
viewModel.ViewDataContext }));
if (Path.GetExtension(viewModel.DataFilePath) == ".aiq") //Load .aiq file format.
{
// Can ignore the returned string since this code is used on project startup, no need to print to
status bar
((WaveformConverterControls.MainWindowViewModel)viewModel.ViewDataContext).OpenAiqFile
CommandExecute(viewModel.DataFilePath);
}
else //Must be .mtwf2 file format.
{
// Can ignore the returned string since this code is used on project startup, no need to print to
status bar
((WaveformConverterControls.MainWindowViewModel)viewModel.ViewDataContext).loadMtwf(vie
wModel.DataFilePath);
}
```

```
pane.Content = viewModel.ContentUserControl;
pane.CanFloat = false; // Floating WindowsFormsHost causes MSChart to disappear, so we disable floating here.

//Line below sets a new host for the window/pane undocked. Problem is knowing when its docked/undocked and where. Maybe have one host each for both states.
//pane.Content = new System.Windows.Forms.Integration.WindowsFormsHost() { Child = viewModel.ContentUserControl };
}
else if (viewModel.ContentType != null)
{
viewModel.ContentUserControl =
(UserControl)Activator.CreateInstance(viewModel.ContentType);
if ((viewModel.ContentUserControl as UserControl).DataContext is IPVMLink)
{
((viewModel.ContentUserControl as UserControl).DataContext as IPVMLink).PVM = viewModel;
}
pane.Content = viewModel.ContentUserControl;
}
return pane;
}
}
else if (item is PVM) //New Code....
{
var viewModel = item as PVM;
if (viewModel != null)
{
var pane = viewModel.IsDocument ? new RadDocumentPane() : new RadPane();
pane.DataContext = item;
RadDocking.SetSerializationTag(pane, viewModel.Header);
if (viewModel.ContentType != null)
{
pane.Content = (UserControl)Activator.CreateInstance(viewModel.ContentType, new object[] { viewModel });
//(pane.Content as UserControl).DataContext = viewModel;
}
return pane;
}
}

return base.CreatePaneForItem(item);
}
```

```
protected override void RemovePane(RadPane pane)
{
///Cannot bring back commented code below. Didn't help much anyway.
///Causes Null Exception when: loading file, closing project with panes in document host, reloading project and then opening any file.
///Presumably due to duplicate panes still being held in memory.
//var group = pane.PaneGroup;
//if (group != null)
//{
//    group.Items.Remove(pane);
//}

pane.Header = null;
pane.DataContext = null;
pane.Content = null;
pane.ClearValue(RadDocking.SerializationTagProperty);
pane.RemoveFromParent();

base.RemovePane(pane); //Only misses the tag clearing thing

pane = null;

GC.Collect();
}
}
}
```

CustomKeyboardCommandProvider.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public class CustomKeyboardCommandProvider : DefaultKeyboardCommandProvider
{
```

```
private GridViewDataControl parentGrid;

public CustomKeyboardCommandProvider(GridViewDataControl grid)
: base(grid)
{
this.parentGrid = grid;
}

public override IEnumerable<ICommand> ProvideCommandsForKey(Key key)
{
List<ICommand> commandsToExecute = base.ProvideCommandsForKey(key).ToList();

if (key == Key.Enter)
{
commandsToExecute.Clear();
commandsToExecute.Add(RadGridViewCommands.CommitEdit);
commandsToExecute.Add(RadGridViewCommands.MoveDown);
//commandsToExecute.Add(RadGridViewCommands.BeginEdit);
}

return commandsToExecute;
}
}
}
```

CustomSaveLoadLayoutHelper.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Docking;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public class CustomSaveLoadLayoutHelper : DefaultSaveLoadLayoutHelper
{
protected override void ElementLoadedOverride(string serializationTag, DependencyObject
element)
{
```

```
base.ElementLoadedOverride(serializationTag, element);
var document = element as RadDocumentPane;
if (document != null)
{
// Restore the content for the loaded document.
}
}
}
}
```

DropIndicationDetails.cs

```
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik
{
public class DropIndicationDetails : ViewModelBase
{
private object currentDraggedItem;
private DropPosition currentDropPosition;
private object currentDraggedOverItem;

public object CurrentDraggedOverItem
{
get
{
return currentDraggedOverItem;
}
set
{
if (this.currentDraggedOverItem != value)
{
currentDraggedOverItem = value;
OnPropertyChanged("CurrentDraggedOverItem");
}
}
}

public int DropIndex { get; set; }

public DropPosition CurrentDropPosition
{
get
```

```
{
return this.currentDropPosition;
}
set
{
if (this.currentDropPosition != value)
{
this.currentDropPosition = value;
OnPropertyChanged("CurrentDropPosition");
}
}
}

public object CurrentDraggedItem
{
get
{
return this.currentDraggedItem;
}
set
{
if (this.currentDraggedItem != value)
{
this.currentDraggedItem = value;
OnPropertyChanged("CurrentDraggedItem");
}
}
}
}
}
```

EnumBindingSourceExtension .cs

```
using System;
using System.Windows.Markup;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public class EnumBindingSourceExtension : MarkupExtension
{
private Type _enumType;
public Type EnumType
{
```

```
get { return this._enumType; }
set
{
if (value != this._enumType)
{
if (null != value)
{
Type enumType = Nullable.GetUnderlyingType(value) ?? value;
if (!enumType.IsEnum)
throw new ArgumentException("Type must be for an Enum.");
}

this._enumType = value;
}
}
}

public EnumBindingSourceExtension() { }

public EnumBindingSourceExtension(Type enumType)
{
this.EnumType = enumType;
}

public override object ProvideValue(IServiceProvider serviceProvider)
{
if (null == this._enumType)
throw new InvalidOperationException("The EnumType must be specified.");

Type actualEnumType = Nullable.GetUnderlyingType(this._enumType) ?? this._enumType;
Array enumValues = Enum.GetValues(actualEnumType);

if (actualEnumType == this._enumType)
return enumValues;

Array tempArray = Array.CreateInstance(actualEnumType, enumValues.Length + 1);
enumValues.CopyTo(tempArray, 1);
return tempArray;
}
}
}
```

EnumDescriptionTypeConverter.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
[Serializable]
public class EnumDescriptionTypeConverter : EnumConverter
{
public EnumDescriptionTypeConverter(Type type)
: base(type)
{
}
public override object ConvertTo(ITypeDescriptorContext context,
System.Globalization.CultureInfo culture, object value, Type destinationType)
{
if (destinationType == typeof(string))
{
if (value != null)
{
FieldInfo fi = value.GetType().GetField(value.ToString());
if (fi != null)
{
var attributes = (DescriptionAttribute[])fi.GetCustomAttributes(typeof(DescriptionAttribute), false);
return ((attributes.Length > 0) && (!String.IsNullOrEmpty(attributes[0].Description))) ?
attributes[0].Description : value.ToString();
}
}

return string.Empty;
}

return base.ConvertTo(context, culture, value, destinationType);
}
}
}
```

GridViewSelectionUtilities.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Windows;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public static class GridViewSelectionUtilities
{
private static bool isSyncingSelection;
private static List<Tuple<WeakReference, List<RadGridView>>> collectionToGridViews = new
List<Tuple<WeakReference, List<RadGridView>>>();

public static readonly DependencyProperty SelectedItemsProperty =
DependencyProperty.RegisterAttached(
"SelectedItems",
typeof(INotifyCollectionChanged),
typeof(GridViewSelectionUtilities),
new PropertyMetadata(null, OnSelectedItemsChanged));

public static INotifyCollectionChanged GetSelectedItems(DependencyObject obj)
{
return (INotifyCollectionChanged)obj.GetValue(SelectedItemsProperty);
}

public static void SetSelectedItems(DependencyObject obj, INotifyCollectionChanged value)
{
obj.SetValue(SelectedItemsProperty, value);
}

private static void OnSelectedItemsChanged(DependencyObject target,
DependencyPropertyChangedEventArgs args)
{
var gridView = (RadGridView)target;

var oldCollection = args.OldValue as INotifyCollectionChanged;
if (oldCollection != null)
{
gridView.SelectionChanged -= GridView_SelectionChanged;
oldCollection.CollectionChanged -= SelectedItems_CollectionChanged;
```

```
RemoveAssociation(oldCollection, gridView);
}

var newCollection = args.NewValue as INotifyCollectionChanged;
if (newCollection != null)
{
gridView.SelectionChanged += GridView_SelectionChanged;
newCollection.CollectionChanged += SelectedItems_CollectionChanged;
AddAssociation(newCollection, gridView);
OnSelectedItemsChanged(newCollection, null, (IList)newCollection);
}
}

private static void SelectedItems_CollectionChanged(object sender,
NotifyCollectionChangedEventArgs args)
{
INotifyCollectionChanged collection = (INotifyCollectionChanged)sender;
OnSelectedItemsChanged(collection, args.OldItems, args.NewItems);
}

private static void GridView_SelectionChanged(object sender, SelectionChangeEventArgs args)
{
if (isSyncingSelection)
{
return;
}

var collection = (IList)GetSelectedItems((RadGridView)sender);
foreach (object item in args.RemovedItems)
{
collection.Remove(item);
}
foreach (object item in args.AddedItems)
{
collection.Add(item);
}
}

private static void OnSelectedItemsChanged(INotifyCollectionChanged collection, IList oldItems,
IList newItems)
{
isSyncingSelection = true;
```

```
var gridViews = GetOrCreateGridViews(collection);
foreach (var gridView in gridViews)
{
SyncSelection(gridView, oldItems, newItems);
}

isSyncingSelection = false;
}

private static void SyncSelection(RadGridView gridView, IList oldItems, IList newItems)
{
if (oldItems != null)
{
SetItemsSelection(gridView, oldItems, false);
}

if (newItems != null)
{
SetItemsSelection(gridView, newItems, true);
}
}

private static void SetItemsSelection(RadGridView gridView, IList items, bool shouldSelect)
{
foreach (var item in items)
{
bool contains = gridView.SelectedItems.Contains(item);
if (shouldSelect && !contains)
{
gridView.SelectedItems.Add(item);
}
else if (contains && !shouldSelect)
{
gridView.SelectedItems.Remove(item);
}
}
}

private static void AddAssociation(INotifyCollectionChanged collection, RadGridView gridView)
{
List<RadGridView> gridViews = GetOrCreateGridViews(collection);
```

```
gridViews.Add(gridView);
}

private static void RemoveAssociation(INotifyCollectionChanged collection, RadGridView
gridView)
{
List<RadGridView> gridViews = GetOrCreateGridViews(collection);
gridViews.Remove(gridView);

if (gridViews.Count == 0)
{
CleanUp();
}
}

private static List<RadGridView> GetOrCreateGridViews(INotifyCollectionChanged collection)
{
for (int i = 0; i < collectionToGridViews.Count; i++)
{
var wr = collectionToGridViews[i].Item1;
if (wr.Target == collection)
{
return collectionToGridViews[i].Item2;
}
}

collectionToGridViews.Add(new Tuple<WeakReference, List<RadGridView>>(new
WeakReference(collection), new List<RadGridView>()));
return collectionToGridViews[collectionToGridViews.Count - 1].Item2;
}

private static void CleanUp()
{
for (int i = collectionToGridViews.Count - 1; i >= 0; i--)
{
bool isAlive = collectionToGridViews[i].Item1.IsAlive;
var behaviors = collectionToGridViews[i].Item2;
if (behaviors.Count == 0 || !isAlive)
{
collectionToGridViews.RemoveAt(i);
}
}
```

```
}
}
}
```

HistoryManager.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Data;
using System.Diagnostics;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{

[Serializable]
public class DataHistoryManager
{
private List<HistoryRecord> historyLog;
private ObservableCollection<object> sourceData;
private bool undoMode;

public DataHistoryManager(ObservableCollection<object> source)
{
historyLog = new List<HistoryRecord>();
sourceData = source;
}

public void Undo()
{
if (historyLog.Count == 0)
{
return;
}
undoMode = true;

HistoryRecord current = historyLog[historyLog.Count - 1];
historyLog.RemoveAt(historyLog.Count - 1);
```

```
PerformUndoAction(current);

undoMode = false;
}

private void PerformUndoAction(HistoryRecord current)
{
switch (current.Action)
{
case HistoryAction.Delete:
if(current.DataItem is List<KeyValuePair<int, object>>)
{
List<KeyValuePair<int, object>> items = new List<KeyValuePair<int, object>>((current.DataItem
as List<KeyValuePair<int, object>>).OrderBy(x => x.Key));
for (int i = 0; i < items.Count; i++)
{
sourceData.Insert(items[i].Key, items[i].Value);
}
}
break;
case HistoryAction.Add:
sourceData.Remove(current.DataItem);
break;
case HistoryAction.Duplicate:
if (current.DataItem is List<KeyValuePair<int, object>>)
{
List<KeyValuePair<int, object>> items = new List<KeyValuePair<int, object>>((current.DataItem
as List<KeyValuePair<int, object>>).OrderByDescending(x => x.Key));
for (int i = 0; i < items.Count; i++)
{
sourceData.Remove(items[i].Value);
}
}
break;
case HistoryAction.Edit:
if (current.DataItem is List<Edit<int, object, string>>)
{
List<Edit<int, object, string>> editedItems = current.DataItem as List<Edit<int, object, string>>;
foreach (Edit<int, object, string> edit in editedItems)
{
object dataItem = sourceData[edit.Index];
```

```
PropertyInfo cellProp = dataItem.GetType().GetProperty(edit.Property);
cellProp.SetValue(dataItem, edit.Value);
}
}
break;
default:
break;
}
}

private void LogAction(HistoryRecord record)
{
if (!undoMode)
{
historyLog.Add(record);
}
}

public void LogEditAction(List<Edit<int, object, string>> editedItems)
{
LogAction(new HistoryRecord()
{
Action = HistoryAction.Edit,
DataItem = editedItems
});
}

public void LogInsertAction(object addedDataItem)
{
LogAction(new HistoryRecord()
{
Action = HistoryAction.Add,
DataItem = addedDataItem
});
}

public void LogDuplicateAction(List<KeyValuePair<int, object>> duplicatedItems)
{
LogAction(new HistoryRecord()
{
Action = HistoryAction.Duplicate,
DataItem = duplicatedItems
```

```
});
}

public void LogDeleteAction(List<KeyValuePair<int, object>> deletedItems)
{
LogAction(new HistoryRecord()
{
Action = HistoryAction.Delete,
DataItem = deletedItems
});
}
}
}
```

MaximizeExtensions.cs

```
using System.Windows;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public static class MaximizeExtensions
{
public static RadPaneGroup GetLastUsedGroup(DependencyObject obj)
{
return (RadPaneGroup)obj.GetValue(LastUsedGroupProperty);
}

public static void SetLastUsedGroup(DependencyObject obj, RadPaneGroup value)
{
obj.SetValue(LastUsedGroupProperty, value);
}

public static readonly DependencyProperty LastUsedGroupProperty =
DependencyProperty.RegisterAttached("LastUsedGroup", typeof(RadPaneGroup),
typeof(MaximizeExtensions), new PropertyMetadata(null));
}
}
```

RowNumberColumn.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
```

```
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public class RowNumberColumn : Telerik.Windows.Controls.GridViewColumn
{
public override FrameworkElement
CreateCellElement(Telerik.Windows.Controls.GridView.GridViewCell cell, object dataItem)
{
TextBlock textBlock = cell.Content as TextBlock;

if (textBlock == null)
{
textBlock = new TextBlock();
}

textBlock.Text = string.Format("{0}", this.DataControl.Items.IndexOf(dataItem) + 1);
textBlock.MouseLeftButtonUp += TextBlock_MouseLeftButtonDown; //When does it
unsubscribe???? Memory Issue.



return textBlock;
}
private void TextBlock_MouseLeftButtonDown(object sender,
System.Windows.Input.MouseButtonEventArgs e)
{
TextBlock textBlock = sender as TextBlock;
GridViewRow row = textBlock.ParentOfType<GridViewRow>();
row.IsSelected = true;
}

protected override void
OnPropertyChanged(System.ComponentModel.PropertyChangedEventArgs args)
{
base.OnPropertyChanged(args);

if (args.PropertyName == "DataControl")
{
if (this.DataControl != null && this.DataControl.Items != null)
{
this.DataControl.Items.CollectionChanged += (s, e) =>
```

```
{
if (e.Action == System.Collections.Specialized.NotifyCollectionChangedAction.Remove)
{
this.Refresh();
}
};
}
}
}

}

}
```

RowReorderBehavior.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
using Telerik.Windows.DragDrop;
using Telerik.Windows.DragDrop.Behaviors;

namespace MerlinTestStudio_Demo_Telerik
{
public class RowReorderBehavior
{
private const string DropPositionFeedbackElementName = "DragBetweenItemsFeedback";
private ContentPresenter dropPositionFeedbackPresenter;
private Grid dropPositionFeedbackPresenterHost;

private RadGridView associatedObject;

/// <summary>
/// AssociatedObject Property
/// </summary>
public RadGridView AssociatedObject
```

```
{
get
{
return associatedObject;
}
set
{
associatedObject = value;
}
}

private static Dictionary<RadGridView, RowReorderBehavior> instances;

static RowReorderBehavior()
{
instances = new Dictionary<RadGridView, RowReorderBehavior>();
}

public static bool GetIsEnabled(DependencyObject obj)
{
return (bool)obj.GetValue(IsEnabledProperty);
}

public static void SetIsEnabled(DependencyObject obj, bool value)
{
RowReorderBehavior behavior = GetAttachedBehavior(obj as RadGridView);

behavior.AssociatedObject = obj as RadGridView;

if (value)
{
behavior.Initialize();
}
else
{
behavior.CleanUp();
}
obj.SetValue(IsEnabledProperty, value);
}

// Using a DependencyProperty as the backing store for IsEnabled.  This enables animation,
styling, binding, etc...
```

```
public static readonly DependencyProperty IsEnabledProperty =
DependencyProperty.RegisterAttached("IsEnabled", typeof(bool), typeof(RowReorderBehavior),
new PropertyMetadata(new PropertyChangedCallback(OnIsEnabledPropertyChanged)));

public static void OnIsEnabledPropertyChanged(DependencyObject dependencyObject,
DependencyPropertyChangedEventArgs e)
{
SetIsEnabled(dependencyObject, (bool)e.NewValue);
}

private static RowReorderBehavior GetAttachedBehavior(RadGridView gridview)
{
if (!instances.ContainsKey(gridview))
{
instances[gridview] = new RowReorderBehavior();
instances[gridview].AssociatedObject = gridview;
}

return instances[gridview];
}

public RowReorderBehavior()
{

}

protected virtual void Initialize()
{
this.AssociatedObject.RowLoaded -= this.AssociatedObject_RowLoaded;
this.AssociatedObject.RowLoaded += this.AssociatedObject_RowLoaded;
this.UnsubscribeFromDragDropEvents();
this.SubscribeToDragDropEvents();

this.AssociatedObject.Dispatcher.BeginInvoke((Action)(() =>
{
this.dropPositionFeedbackPresenter = new ContentPresenter();
this.dropPositionFeedbackPresenter.Name = DropPositionFeedbackElementName;
this.dropPositionFeedbackPresenter.HorizontalAlignment = HorizontalAlignment.Left;
this.dropPositionFeedbackPresenter.VerticalAlignment = VerticalAlignment.Top;
this.dropPositionFeedbackPresenter.RenderTransformOrigin = new Point(0.5, 0.5);

this.AttachDropPositionFeedback();
```

```
}));
}

protected virtual void CleanUp()
{
this.AssociatedObject.RowLoaded -= this.AssociatedObject_RowLoaded;
this.UnsubscribeFromDragDropEvents();

this.DetachDropPositionFeedback();
}

void AssociatedObject_RowLoaded(object sender,
Telerik.Windows.Controls.GridView.RowLoadedEventArgs e)
{
if (e.Row is GridViewHeaderRow || e.Row is GridViewNewRow || e.Row is GridViewFooterRow)
return;

GridViewRow row = e.Row as GridViewRow;
this.InitializeRowDragAndDrop(row);
}

private void InitializeRowDragAndDrop(GridViewRow row)
{
if (row == null)
return;

DragDropManager.RemoveDragOverHandler(row, OnRowDragOver);
DragDropManager.AddDragOverHandler(row, OnRowDragOver);
}

private void SubscribeToDragDropEvents()
{
DragDropManager.AddDragInitializeHandler(this.AssociatedObject, OnDragInitialize);
DragDropManager.AddGiveFeedbackHandler(this.AssociatedObject, OnGiveFeedback);
DragDropManager.AddDropHandler(this.AssociatedObject, OnDrop);
DragDropManager.AddDragDropCompletedHandler(this.AssociatedObject,
OnDragDropCompleted);
}

private void UnsubscribeFromDragDropEvents()
{
DragDropManager.RemoveDragInitializeHandler(this.AssociatedObject, OnDragInitialize);
```

```
DragDropManager.RemoveGiveFeedbackHandler(this.AssociatedObject, OnGiveFeedback);
DragDropManager.RemoveDropHandler(this.AssociatedObject, OnDrop);
DragDropManager.RemoveDragDropCompletedHandler(this.AssociatedObject,
OnDragDropCompleted);
}

private void OnDragDropCompleted(object sender, DragDropCompletedEventArgs e)
{
this.HideDropPositionFeedbackPresenter();
}

private void OnDragInitialize(object sender, DragInitializeEventArgs e)
{
var sourceRow = e.OriginalSource as GridViewRow ?? (e.OriginalSource as
FrameworkElement).ParentOfType<GridViewRow>();
if (sourceRow != null && sourceRow.Name != "PART_RowResizer")
{
DropIndicationDetails details = new DropIndicationDetails();
var item = sourceRow.Item;
details.CurrentDraggedItem = item;

IDragPayload dragPayload = DragDropPayloadManager.GeneratePayload(null);

dragPayload.SetData("DraggedItem", item);
dragPayload.SetData("DropDetails", details);

e.Data = dragPayload;

e.DragVisual = new DragVisual()
{
Content = details,
ContentTemplate = this.AssociatedObject.Resources["DraggedItemTemplate"] as DataTemplate
};
e.DragVisualOffset = e.RelativeStartPoint;
e.AllowedEffects = DragDropEffects.All;
}
}

private void OnGiveFeedback(object sender, Telerik.Windows.DragDrop.GiveFeedbackEventArgs
e)
{
e.SetCursor(Cursors.Arrow);
```

```
e.Handled = true;
}

private void OnDrop(object sender, Telerik.Windows.DragDrop.DragEventArgs e)
{
var draggedItem = DragDropPayloadManager.GetDataFromObject(e.Data, "DraggedItem");
var details = DragDropPayloadManager.GetDataFromObject(e.Data, "DropDetails") as
DropIndicationDetails;

if (details == null || draggedItem == null)
{
return;
}

if (e.Effects == DragDropEffects.Move || e.Effects == DragDropEffects.All)
{
((sender as RadGridView).ItemsSource as IList).Remove(draggedItem);
}

if (e.Effects != DragDropEffects.None)
{
var collection = (sender as RadGridView).ItemsSource as IList;
int index = details.DropIndex < 0 ? 0 : details.DropIndex;
index = details.DropIndex > collection.Count - 1 ? collection.Count : index;

collection.Insert(index, draggedItem);
}

HideDropPositionFeedbackPresenter();
}

private void OnRowDragOver(object sender, Telerik.Windows.DragDrop.DragEventArgs e)
{
var row = sender as GridViewRow;
var details = DragDropPayloadManager.GetDataFromObject(e.Data, "DropDetails") as
DropIndicationDetails;

if (details == null || row == null)
{
return;
}
```

```
details.CurrentDraggedOverItem = row.DataContext;

if (details.CurrentDraggedItem == details.CurrentDraggedOverItem)
{
e.Effects = DragDropEffects.None;
e.Handled = true;
return;
}

details.CurrentDropPosition = GetDropPositionFromPoint(e.GetPosition(row), row);
int dropIndex = (this.AssociatedObject.Items as IList).IndexOf(row.DataContext);
int draggedItemIdex = (this.AssociatedObject.Items as
IList).IndexOf(DragDropPayloadManager.GetDataFromObject(e.Data, "DraggedItem"));

if (dropIndex >= row.GridViewDataControl.Items.Count - 1 && details.CurrentDropPosition ==
DropPosition.After)
{
details.DropIndex = dropIndex;
this.ShowDropPositionFeedbackPresenter(this.AssociatedObject, row,
details.CurrentDropPosition);
return;
}

dropIndex = draggedItemIdex > dropIndex ? dropIndex : dropIndex - 1;
details.DropIndex = details.CurrentDropPosition == DropPosition.Before ? dropIndex : dropIndex +
1;

this.ShowDropPositionFeedbackPresenter(this.AssociatedObject, row,
details.CurrentDropPosition);
}

public virtual DropPosition GetDropPositionFromPoint(Point absoluteMousePosition,
GridViewRow row)
{
if (row != null)
{
return absoluteMousePosition.Y < row.ActualHeight / 2 ? DropPosition.Before : DropPosition.After;
}

return DropPosition.Inside;
}
```

```
private bool IsDropPositionFeedbackAvailable()
{
return
this.dropPositionFeedbackPresenterHost != null &&
this.dropPositionFeedbackPresenter != null;
}

private void ShowDropPositionFeedbackPresenter(GridViewDataControl gridView, GridViewRow
row, DropPosition lastRowDropPosition)
{
if (!this.IsDropPositionFeedbackAvailable())
return;
var yOffset = this.GetDropPositionFeedbackOffset(row, lastRowDropPosition);
this.dropPositionFeedbackPresenter.Visibility = Visibility.Visible;
this.dropPositionFeedbackPresenter.Width = row.ActualWidth;
this.dropPositionFeedbackPresenter.RenderTransform = new TranslateTransform()
{
Y = yOffset
};
}

private void HideDropPositionFeedbackPresenter()
{
this.dropPositionFeedbackPresenter.RenderTransform = new TranslateTransform()
{
X = 0,
Y = 0
};
this.dropPositionFeedbackPresenter.Visibility = Visibility.Collapsed;
}

private double GetDropPositionFeedbackOffset(GridViewRow row, DropPosition dropPosition)
{
var yOffset = row.TransformToVisual(this.dropPositionFeedbackPresenterHost).Transform(new
Point(0, 0)).Y;
if (dropPosition == DropPosition.After)
yOffset += row.ActualHeight;
yOffset -= (this.dropPositionFeedbackPresenter.ActualHeight / 2.0);
return yOffset;
}

private void DetachDropPositionFeedback()
```

```
{
if (this.IsDropPositionFeedbackAvailable())
{
this.dropPositionFeedbackPresenterHost.Children.Remove(this.dropPositionFeedbackPresenter);
this.dropPositionFeedbackPresenter = null;
}
}

private void AttachDropPositionFeedback()
{
this.dropPositionFeedbackPresenterHost = this.AssociatedObject.ParentOfType<Grid>();

if (this.dropPositionFeedbackPresenterHost != null)
{
this.dropPositionFeedbackPresenter.Content = CreateDefaultDropPositionFeedback();
if (dropPositionFeedbackPresenterHost != null &&
dropPositionFeedbackPresenterHost.FindName(this.dropPositionFeedbackPresenter.Name) ==
null)
{
this.dropPositionFeedbackPresenterHost.Children.Add(this.dropPositionFeedbackPresenter);
}
}
this.HideDropPositionFeedbackPresenter();
}

private static UIElement CreateDefaultDropPositionFeedback()
{
Grid grid = new Grid()
{
Height = 8,
HorizontalAlignment = HorizontalAlignment.Stretch,
IsHitTestVisible = false,
VerticalAlignment = VerticalAlignment.Stretch
};
grid.ColumnDefinitions.Add(new ColumnDefinition()
{
Width = new GridLength(8)
});
grid.ColumnDefinitions.Add(new ColumnDefinition());
Ellipse ellipse = new Ellipse()
{
Stroke = new SolidColorBrush(Colors.Orange),
```

```
StrokeThickness = 2,
Fill = new SolidColorBrush(Colors.Orange),
HorizontalAlignment = HorizontalAlignment.Stretch,
VerticalAlignment = VerticalAlignment.Stretch,
Width = 8,
Height = 8
};
Rectangle rectangle = new Rectangle()
{
Fill = new SolidColorBrush(Colors.Orange),
RadiusX = 2,
RadiusY = 2,
VerticalAlignment = VerticalAlignment.Stretch,
HorizontalAlignment = HorizontalAlignment.Stretch,
Height = 2
};
Grid.SetColumn(ellipse, 0);
Grid.SetColumn(rectangle, 1);
grid.Children.Add(ellipse);
grid.Children.Add(rectangle);

Canvas.SetZIndex(grid, 10000);

return grid;
}
}
}
```

TypeMerger.cs

```
using System;
using System.Collections.Generic;
using System.Reflection;
using System.Reflection.Emit;
using System.ComponentModel;
using System.Linq;
using System.Linq.Expressions;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
/// <summary>
/// A Utility class used to merge the properties of heterogenious objects
/// </summary>
```

```
public static class TypeMerger {

//assembly/module builders
private static AssemblyBuilder asmBuilder = null;
private static ModuleBuilder modBuilder = null;
private static TypeMergerPolicy typeMergerPolicy = null;

//object type cache
private static IDictionary<String, Type> anonymousTypes = new Dictionary<String, Type>();

//used for thread-safe access to Type Dictionary
private static Object _syncLock = new Object();

/// <summary>
/// Merge two different object instances into a single object which is a super-set of the properties of both objects.
/// If property name collision occurs and no policy has been created to specify which to use using the .Use() method the property value from 'values1' will be used.
/// </summary>
/// <param name="values1">An object to be merged.</param>
/// <param name="values2">An object to be merged.</param>
/// <returns>New object containing properties from both objects</returns>
public static Object Merge(Object values1, Object values2) {

//lock for thread safe writing
lock (_syncLock) {

//create a name from the names of both Types
var name = String.Format("{0}_{1}", values1.GetType(),
values2.GetType());

if (typeMergerPolicy != null) {
name += "_" + string.Join(",", typeMergerPolicy.IgnoredProperties.Select(x =>
String.Format("{0}_{1}", x.Item1, x.Item2)));
}

//now that we're inside the lock - check one more time
var result = CreateInstance(name, values1, values2);
if (result != null) {
typeMergerPolicy = null;
return result;
}
```

```
//merge list of PropertyDescriptors for both objects
var pdc = GetProperties(values1, values2);

//make sure static properties are properly initialized
InitializeAssembly();

//create the type definition
var newType = CreateType(name, pdc);

//add it to the cache
anonymousTypes.Add(name, newType);

//return an instance of the new Type
result = CreateInstance(name, values1, values2);
typeMergerPolicy = null;
return result;
}
}

/// <summary>
/// Used internally by the TypeMergerPolicy class method chaining for specifying a policy to use during merging.
/// </summary>
internal static Object Merge(object values1, object values2, TypeMergerPolicy policy) {

typeMergerPolicy = policy;
return Merge(values1, values2);
}

/// <summary>
/// Specify a property to be ignored from a object being merged.
/// </summary>
/// <param name="ignoreProperty">The property of the object to be ignored as a Func.</param>
/// <returns>TypeMerger policy used in method chaining.</returns>
public static TypeMergerPolicy Ignore(Expression<Func<object>> ignoreProperty) {
return new TypeMergerPolicy().Ignore(ignoreProperty);
}

/// <summary>
/// Specify a property to use when there is a property name collision between objects being merged.
```

```
/// </summary>
/// <param name="useProperty"></param>
/// <returns>TypeMerger policy used in method chaining.</returns>
public static TypeMergerPolicy Use(Expression<Func<object>> useProperty) {
return new TypeMergerPolicy().Use(useProperty);
}

/// <summary>
/// Instantiates an instance of an existing Type from cache
/// </summary>
private static Object CreateInstance(String name, object values1, object values2) {

Object newValues = null;

//check to see if type exists
if (anonymousTypes.ContainsKey(name)) {

//merge all values together into an array
var allValues = MergeValues(values1, values2);

//get type
var type = anonymousTypes[name];

//make sure it isn't null for some reason
if (type != null) {
//create a new instance
newValues = Activator.CreateInstance(type, allValues);
} else {
//remove null type entry
lock (_syncLock)
anonymousTypes.Remove(name);
}
}

//return values (if any)
return newValues;
}

/// <summary>
/// Merge PropertyDescriptors for both objects
/// </summary>
private static PropertyDescriptor[] GetProperties(object values1, object values2) {
```

```
//dynamic list to hold merged list of properties
var properties = new List<PropertyDescriptor>();

//get the properties from both objects
var pdc1 = TypeDescriptor.GetProperties(values1);
var pdc2 = TypeDescriptor.GetProperties(values2);

//add properties from values1
for (int i = 0; i < pdc1.Count; i++) {
if (typeMergerPolicy == null
|| (!typeMergerPolicy.IgnoredProperties.Contains(new Tuple<string,
string>(values1.GetType().Name, pdc1[i].Name))
& !typeMergerPolicy.UseProperties.Contains(new Tuple<string, string>(values2.GetType().Name,
pdc1[i].Name))))
properties.Add(pdc1[i]);
}
//add properties from values2
for (int i = 0; i < pdc2.Count; i++) {
if (typeMergerPolicy == null
|| (!typeMergerPolicy.IgnoredProperties.Contains(new Tuple<string,
string>(values2.GetType().Name, pdc2[i].Name))
& !typeMergerPolicy.UseProperties.Contains(new Tuple<string, string>(values1.GetType().Name,
pdc2[i].Name))))
properties.Add(pdc2[i]);
}
//return array
return properties.ToArray();
}

/// <summary>
/// Get the type of each property
/// </summary>
private static Type[] GetTypes(PropertyDescriptor[] pdc) {

var types = new List<Type>();

for (int i = 0; i < pdc.Length; i++)
types.Add(pdc[i].PropertyType);

return types.ToArray();
}
```

```
/// <summary>
/// Merge the values of the two types into an object array
/// </summary>
private static Object[] MergeValues(object values1, object values2) {

var pdc1 = TypeDescriptor.GetProperties(values1);
var pdc2 = TypeDescriptor.GetProperties(values2);

var values = new List<Object>();
for (int i = 0; i < pdc1.Count; i++) {
if (typeMergerPolicy == null
|| (!typeMergerPolicy.IgnoredProperties.Contains(new Tuple<string,
string>(values1.GetType().Name, pdc1[i].Name))
& !typeMergerPolicy.UseProperties.Contains(new Tuple<string, string>(values2.GetType().Name,
pdc1[i].Name))))
values.Add(pdc1[i].GetValue(values1));
}

for (int i = 0; i < pdc2.Count; i++) {
if (typeMergerPolicy == null
|| (!typeMergerPolicy.IgnoredProperties.Contains(new Tuple<string,
string>(values2.GetType().Name, pdc2[i].Name))
& !typeMergerPolicy.UseProperties.Contains(new Tuple<string, string>(values1.GetType().Name,
pdc2[i].Name))))
values.Add(pdc2[i].GetValue(values2));
}
return values.ToArray();
}

/// <summary>
/// Initialize static objects
/// </summary>
private static void InitializeAssembly() {

//check to see if we've already instantiated
//the static objects
if (asmBuilder == null) {
//create a new dynamic assembly
var assembly = new AssemblyName();
assembly.Name = "AnonymousTypeExentions";
```

```
//get a module builder object
asmBuilder = AssemblyBuilder.DefineDynamicAssembly(assembly, AssemblyBuilderAccess.Run);
modBuilder = asmBuilder.DefineDynamicModule(asmBuilder.GetName().Name);
}
}

/// <summary>
/// Create a new Type definition from the list
/// of PropertyDescriptors
/// </summary>
private static Type CreateType(String name, PropertyDescriptor[] pdc) {

//create TypeBuilder
var typeBuilder = CreateTypeBuilder(name);

//get list of types for ctor definition
var types = GetTypes(pdc);

//create priate fields for use w/in the ctor body and properties
var fields = BuildFields(typeBuilder, pdc);

//define/emit the Ctor
BuildCtor(typeBuilder, fields, types);

//define/emit the properties
BuildProperties(typeBuilder, fields);

//return Type definition
return typeBuilder.CreateType();
}

/// <summary>
/// Create a type builder with the specified name
/// </summary>
private static TypeBuilder CreateTypeBuilder(string typeName) {

var typeBuilder = modBuilder.DefineType(typeName,
TypeAttributes.Public,
typeof(object));
//return new type builder
return typeBuilder;
}
```

```
/// <summary>
/// Define/emit the ctor and ctor body
/// </summary>
private static void BuildCtor(TypeBuilder typeBuilder, FieldBuilder[] fields, Type[] types) {

//define ctor()
var ctor = typeBuilder.DefineConstructor(
MethodAttributes.Public,
CallingConventions.Standard,
types
);

//build ctor()
var ctorGen = ctor.GetILGenerator();

//create ctor that will assign to private fields
for (int i = 0; i < fields.Length; i++) {
//load argument (parameter)
ctorGen.Emit(OpCodes.Ldarg_0);
ctorGen.Emit(OpCodes.Ldarg, (i + 1));

//store argument in field
ctorGen.Emit(OpCodes.Stfld, fields[i]);
}

//return from ctor()
ctorGen.Emit(OpCodes.Ret);
}

/// <summary>
/// Define fields based on the list of PropertyDescriptors
/// </summary>
private static FieldBuilder[] BuildFields(TypeBuilder typeBuilder, PropertyDescriptor[] pdc) {

var fields = new List<FieldBuilder>();

//build/define fields
for (int i = 0; i < pdc.Length; i++) {
var pd = pdc[i];

//define field as '_[Name]' with the object's Type
```

```
var field = typeBuilder.DefineField(
String.Format("_{0}", pd.Name),
pd.PropertyType,
FieldAttributes.Private
);

//add to list of FieldBuilder objects
if (fields.Contains(field) == false)
fields.Add(field);
}

return fields.ToArray();
}

/// <summary>
/// Build a list of Properties to match the list of private fields
/// </summary>
private static void BuildProperties(TypeBuilder typeBuilder, FieldBuilder[] fields) {

//build properties
for (int i = 0; i < fields.Length; i++) {
//remove '_' from name for public property name
var propertyName = fields[i].Name.Substring(1);

//define the property
var property = typeBuilder.DefineProperty(propertyName,
PropertyAttributes.None, fields[i].FieldType, null);

//define 'Get' method only (anonymous types are read-only)
var getMethod = typeBuilder.DefineMethod(
String.Format("Get_{0}", propertyName),
MethodAttributes.Public,
fields[i].FieldType,
Type.EmptyTypes
);

//build 'Get' method
var methGen = getMethod.GetILGenerator();

//method body
methGen.Emit(OpCodes.Ldarg_0);
//load value of corresponding field
```

```
methGen.Emit(OpCodes.Ldfld, fields[i]);
//return from 'Get' method
methGen.Emit(OpCodes.Ret);

//assign method to property 'Get'
property.SetGetMethod(getMethod);

//TODO: look into this....
////define 'Set' method
//MethodBuilder setMethod = typeBuilder.DefineMethod(
//    String.Format("Set_{0}", propertyName),
//    MethodAttributes.Public | MethodAttributes.SpecialName,
//    null,
//    new Type[] { fields[i].FieldType }
//);

//ILGenerator setMethodGenerator = setMethod.GetILGenerator();
//setMethodGenerator.Emit(OpCodes.Ldarg_0);     // load this
//setMethodGenerator.Emit(OpCodes.Ldarg_1);     // load value
//setMethodGenerator.Emit(OpCodes.Stfld, fields[i]); // store into field
//setMethodGenerator.Emit(OpCodes.Ret); // return

//property.SetSetMethod(setMethod);

}
}
}
}
```

TypeMergerPolicy.cs

```
using System;
using System.Collections.Generic;
using System.Linq.Expressions;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
/// <summary>
/// Policy class used with the TypeMerger class to define properties that are ignored or priority when there objects contain the same properties and there is a collision.
/// </summary>
public class TypeMergerPolicy {
```

```
private IList<Tuple<string, string>> ignoredProperties;
private IList<Tuple<string, string>> useProperties;

public TypeMergerPolicy() {

ignoredProperties = new List<Tuple<string, string>>();
useProperties = new List<Tuple<string, string>>();
}

internal IList<Tuple<string, string>> IgnoredProperties {

get { return this.ignoredProperties; }
}

internal IList<Tuple<string, string>> UseProperties {

get { return this.useProperties; }
}

/// <summary>
/// Specify a property to be ignored from a object being merged.
/// </summary>
/// <param name="ignoreProperty">The property of the object to be ignored as a Func.</param>
/// <returns>TypeMerger policy used in method chaining.</returns>
public TypeMergerPolicy Ignore(Expression<Func<object>> ignoreProperty) {

ignoredProperties.Add(GetObjectTypeAndProperty(ignoreProperty));
return this;
}

/// <summary>
/// Specify a property to use when there is a property name collision between objects being
merged.
/// </summary>
/// <param name="useProperty"></param>
/// <returns>TypeMerger policy used in method chaining.</returns>
public TypeMergerPolicy Use(Expression<Func<object>> useProperty) {

useProperties.Add(GetObjectTypeAndProperty(useProperty));
return this;
}
```

```
/// <summary>
///   /// Merge two different object instances into a single object which is a super-set of the properties of both objects.
/// If property name collision occurs and no policy has been created to specify which to use using the .Use() method the property value from 'values1' will be used.
/// </summary>
/// <param name="values1">An object to be merged.</param>
/// <param name="values2">An object to be merged.</param>
/// <returns>New object containing properties from both objects</returns>
public Object Merge(object values1, object values2) {

return TypeMerger.Merge(values1, values2, this);
}

/// <summary>
/// Inspects the property specified to get the underlying Type and property name to be used during merging.
/// </summary>
/// <param name="property">The property to inspect as a Func Expression.</param>
/// <returns></returns>
private Tuple<string, string> GetObjectTypeAndProperty(Expression<Func<object>> property) {

var objType = string.Empty;
var propName = string.Empty;

try {
if (property.Body is MemberExpression) {
objType = ((MemberExpression)property.Body).Expression.Type.Name;
propName = ((MemberExpression)property.Body).Member.Name;
} else if (property.Body is UnaryExpression) {
objType =
((MemberExpression)((UnaryExpression)property.Body).Operand).Expression.Type.Name;
propName = ((MemberExpression)((UnaryExpression)property.Body).Operand).Member.Name;
} else {
throw new Exception("Expression type unknown.");
}
} catch (Exception ex) {
throw new Exception("Error in TypeMergePolicy.GetObjectTypeAndProperty.", ex);
}

return new Tuple<string, string>(objType, propName);
}
```

```
}
}
```

UnitHelper.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using UnitConversionLib;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public static class UnitHelper
{

public static double ValueUnitModifier(double value, string oldUnit, string newUnit)
{
if (double.IsNaN(value) || double.IsInfinity(value))
{
return value;
}

double modifiedValue = value;
try
{
if (!string.IsNullOrEmpty(oldUnit) && !string.IsNullOrEmpty(newUnit))
{
double ParsableValue = value < 0 ? value * -1 : value; //value needs to go into the
Mesurable.Parse() as a positive number.
double convertedValue = Measurable.Parse($"{ParsableValue}
{oldUnit}").ConvertTo(newUnit).Amount;
modifiedValue = value < 0 ? (convertedValue * -1) : convertedValue;
modifiedValue = Math.Round(modifiedValue, 5, MidpointRounding.AwayFromZero); //Temp.
Solution to 79999.99999
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }

return modifiedValue;
}
```

```
public static void CreateUnits()
{
//Volts
Unit v = Unit.CreateNew("V");
Unit dV = v / 10;
Unit.Register(ref dV, "dV");
Unit cV = dV / 10;
Unit.Register(ref cV, "cV");
Unit mV = cV / 10;
Unit.Register(ref mV, "mV");
Unit uV = mV / 1000;
Unit.Register(ref uV, "uV");
Unit nV = uV / 1000;
Unit.Register(ref nV, "nV");
Unit pV = nV / 1000;
Unit.Register(ref pV, "pV");

//Amps
Unit a = Unit.GetRegisteredUnit("A"); ;
Unit dA = a / 10;
Unit.Register(ref dA, "dA");
Unit cA = dA / 10;
Unit.Register(ref cA, "cA");
Unit mA = cA / 10;
Unit.Register(ref mA, "mA");
Unit uA = mA / 1000;
Unit.Register(ref uA, "uA");
Unit nA = uA / 1000;
Unit.Register(ref nA, "nA");
Unit pA = nA / 1000;
Unit.Register(ref pA, "pA");

//Frequency
Unit Hz = Unit.CreateNew("Hz");
Unit daHz = Hz * 10;
Unit.Register(ref daHz, "daHz");
Unit hHz = daHz * 10;
Unit.Register(ref hHz, "hHz");
Unit KHz = hHz * 10;
Unit.Register(ref KHz, "KHz");
Unit MHz = KHz * 1000;
```

```
Unit.Register(ref MHz, "MHz");
Unit GHz = MHz * 1000;
Unit.Register(ref GHz, "GHz");
Unit THz = GHz * 1000;
Unit.Register(ref THz, "THz");

Unit percentage = Unit.CreateNew("%");
Unit number = Unit.CreateNew("#");
Unit dbm = Unit.CreateNew("dBm");
Unit db = Unit.CreateNew("dB");
}
}
}
```

ValidationErrorManager.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models.AppModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public class ValidationErrorManager
{
private static List<Tuple<Guid, Error>> _errors = new List<Tuple<Guid, Error>>();

public static event Action<Error> ErrorAdded;
public static event Action<Guid> ErrorRemoved;

public static Guid AddError(Error error)
{
Guid errorId = Guid.NewGuid();
_errors.Add(new Tuple<Guid, Error>(errorId, error));

ErrorAdded?.Invoke(error);

return errorId;
}

public static void RemoveError(Guid errorId)
```

```
{
var error = _errors.FirstOrDefault(e => e.Item1 == errorId);
if (error != null)
{
_errors.Remove(error);

ErrorRemoved?.Invoke(errorId);
}
}

public static IEnumerable<Tuple<Guid, Error>> GetErrors()
{
return _errors;
}

public static void ClearErrors()
{
_errors.Clear();
}

}

public class ErrorProvider
{
private List<Guid> _errorIds = new List<Guid>();

public IEnumerable<Guid> ErrorIds
{
get { return _errorIds; }
}

public void AddErrors<T>(T data, Dictionary<Func<T, bool>, string> rules)
{
// Check if there are already errors for the given data
bool hasErrors = _errorIds.Any();
if (!hasErrors)
{
List<Error> errors = Validator.ValidateToErrors(data, rules);
foreach (Error error in errors)
{
Guid errorId = ValidationErrorManager.AddError(error);
_errorIds.Add(errorId);
```

```
}
}
}

public void RemoveError(Guid errorId)
{
ValidationErrorManager.RemoveError(errorId);
_errorIds.Remove(errorId);
}

public void ClearErrors()
{
foreach (Guid errorId in _errorIds)
{
ValidationErrorManager.RemoveError(errorId);
}
_errorIds.Clear();
}

}


//public class ErrorException : Exception
//{
//    public ErrorException(string message) : base(message)
//    {
//    }
//
//    public override string Message
//    {
//        get { return base.Message; }
//    }
//
//    public override string ToString()
//    {
//        return base.ToString();
//    }
//}

}
```

Validator.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models.AppModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Helpers
{
public static class Validator
{
/// Currently used in Pin Map check. Need to replace.
// Validates the given data against a set of rules.
// Returns a list of validation errors, if any.
public static List<string> Validate<T>(T data, Dictionary<Func<T, bool>, string> rules)
{
List<string> errors = new List<string>();

foreach (var rule in rules)
{
if (!rule.Key(data))
{
errors.Add(rule.Value);
}
}

return errors;
}

//Used in Error Manager.
public static List<Error> ValidateToErrors<T>(T data, Dictionary<Func<T, bool>, string> rules)
{
List<Error> results = new List<Error>();

foreach (var rule in rules)
{
if (!rule.Key(data))
{
Error error = new Error
{
Description = rule.Value
};
```

```
results.Add(error);
}
}

return results;
}
}

}
```

DataTag.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using Telerik.Windows.Documents.Spreadsheet.Model;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections.ObjectModel;

namespace MerlinTestStudio_Demo_Telerik.Data
{
public interface IDataTag
{
bool IsAutoGen { get; set; }

string ColumnName { get; set; }

int ColumnIndex { get; set; }

string UnitType { get; set; }

string Unit { get; set; }

bool IsUnitable { get; }

TestAttributeType AttributeType { get; set; }
}

//Test Parameter Attribute Type
public enum TestAttributeType
{
```

```
Test_Number,
Test_Name,
Test_Unit,
Function_Call,
Group,
Pin_Parameter,
Pin,
Pout,
Pout_Tolerance,
Frequency,
Waveform,

Digital_Mode,
Parameter,
Parameter_Unit,

//Spacer.

Pattern_Mode_Name,
Pattern_Mode_Register,
Default
}

//Test Limit Attribute Type
public enum TestLimitAttributeType
{
FTLower,
FTUpper,
QALower,
QAUpper,
GoldRetestLower,
GoldRetestUpper,
GoldRetestControl,
GoldTestLower ,
GoldTestUpper,
GoldTestControl ,
GoldLinearLower,
GoldLinearUpper ,
GoldLinearControl ,
StopOnFail,
ApplyAfter ,
OffsetSite1 ,
```

```
OffsetSite2 ,
OffsetSite3 ,
OffsetSite4,
OffsetSite5,
OffsetSite6,
OffsetSite7,
OffsetSite8,
HardBinNumber ,
HardBinName,
HardBinPF ,
SoftBinNumber ,
SoftBinName ,
SoftBinPF ,
FailPriority,
StopOnFailDefault
}

/// <summary>
/// DataTag is used for referencing the data that needs to be imported in the Data Formatting Tool.
/// </summary>'
[Serializable]
public class DataTag : IDataTag, INotifyPropertyChanged
{
private string columnName;
private TestAttributeType attributeType;
private int columnIndex;
private int rowIndex;
private bool isAutoGen;
private string _unit;
private string unitType;

public DataTag(bool _isAutoGen)
{
isAutoGen = _isAutoGen;
}

public bool IsAutoGen { get { return isAutoGen; } set { isAutoGen = value;
OnPropertyChanged("IsAutoGen"); } }
public string ColumnName
{
get { return columnName; }
set { columnName = value; OnPropertyChanged("ColumnName"); }
```

```
}
public TestAttributeType AttributeType
{
get { return attributeType; }
set { attributeType = value; OnPropertyChanged("AttributeType"); }
}
public int ColumnIndex
{
get { return columnIndex; }
set { columnIndex = value; OnPropertyChanged("ColumnIndex"); }
}
public int RowIndex
{
get { return rowIndex; }
set { rowIndex = value; OnPropertyChanged("RowIndex"); }
}
public CellIndex TagCellIndex
{
get { return new CellIndex(RowIndex, ColumnIndex); }
}
public string UnitType
{
get { return unitType; }
set { unitType = value; OnPropertyChanged("UnitType"); OnPropertyChanged("Units"); }
}
public string Unit
{
get { return _unit; }
set { _unit = value; OnPropertyChanged("Unit"); }
}

public List<string> Units
{
get { return GetUnits(UnitType); }
}

//Needs to be converted to a static list in the "Backend Constants" Class!!!
private List<string> GetUnits(string unitType)
{
var units = new List<string>();
switch (unitType)
{
```

```
case "Volts":
units = new List<string>() { "V", "dV", "cV", "mV", "uV", "nV", "pV" };
break;
case "Amps":
units = new List<string>() { "A", "dA", "cA", "mA", "uA", "nA", "pA" };
break;
case "Frequency":
units = new List<string>() { "THz", "GHz", "MHz", "KHz", "hHz", "daHz", "Hz", "dHz", "cHz", "mHz",
"uHz", "nHz", "pHz" };
break;
case "Power":
units = new List<string>() { "dBm", "dB" };
break;
case "Percentage":
units = new List<string>() { "%" };
break;
case "Number":
units = new List<string>() { "#" };
break;
}
return units;
}

public bool IsUnitable
{
get
{
switch ((int)AttributeType)
{
case 0:
case 1:
case 2:
case 3:
case 4:
case 5:
case 10:
return false;
default:
return true;

}
}
```

```
}

#region INPC
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}

[Serializable]
public class ParameterMapTag : IDataTag, INotifyPropertyChanged
{
private string columnName;
private TestAttributeType attributeType;
private int columnIndex;
private bool isAutoGen;
private string unit;
private string unitType;

public ParameterMapTag()
{

}

public string UnitType
{
get { return unitType; }
set { unitType = value; OnPropertyChanged("UnitType"); OnPropertyChanged("Units"); }
}

public string Unit
{
get { return unit; }
set { unit = value; OnPropertyChanged("Unit"); }
}


#region IDataTag
```

```
public bool IsAutoGen
{
get { return isAutoGen; }
set { isAutoGen = value; OnPropertyChanged("IsAutoGen"); }
}

/// <summary>
/// The column title with the unit attached.
/// </summary>
public string ColumnName
{
get { return columnName; }
set { columnName = value; OnPropertyChanged("ColumnName"); }
}
public TestAttributeType AttributeType
{
get { return attributeType; }
set { attributeType = value; OnPropertyChanged("AttributeType"); }
}
public int ColumnIndex
{
get { return columnIndex; }
set { columnIndex = value; OnPropertyChanged("ColumnIndex"); }
}

public bool IsUnitable
{
get
{
switch ((int)AttributeType)
{
case 0:
case 1:
case 2:
case 3:
case 4:
case 5:
case 10:
return false;
default:
return true;
```

```
}
}
}
#endregion
public List<string> Units
{
get { return GetUnits(UnitType); }
}

private List<string> GetUnits(string unitType)
{
var units = new List<string>();
switch (unitType)
{
case "Volts":
units = new List<string>() { "V", "dV", "cV", "mV", "uV", "nV", "pV" };
break;
case "Amps":
units = new List<string>() { "A", "dA", "cA", "mA", "uA", "nA", "pA" };
break;
case "Frequency":
units = new List<string>() { "THz", "GHz", "MHz", "KHz", "hHz", "daHz", "Hz", "dHz", "cHz", "mHz",
"uHz", "nHz", "pHz" };
break;
case "Power":
units = new List<string>() { "dBm", "dB" };
break;
case "Percentage":
units = new List<string>() { };
break;
case "Number":
units = new List<string>() { };
break;
}
return units;
}

#region INPC
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string propertyName)
{
```

```
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
}
```

DLL.cs

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;

namespace MerlinTestStudio_Demo_Telerik.Data
{
[Serializable]
public class DLL : INotifyPropertyChanged
{
private string _nameOfDLL;
private string _dllFilePath;
private ObservableCollection<Function> _dllFunctions;

/// <summary>
/// Creates a DLL object that can be refrenced later to retrieve the DLL's functions.
/// </summary>
/// <param name="Name">Sets the name of the DLL object.</param>
/// <param name="FilePath">Sets the file path of the DLL object.</param>
public DLL()
{

}

/// <summary>
/// The name of the DLL this object represents.
/// </summary>
public string NameOfDLL
{
get { return _nameOfDLL; }
set { _nameOfDLL = value; OnPropertyChanged("NameOfDLL"); }
}

/// <summary>
/// The file path of the DLL this object represents.
```

```
/// </summary>
public string DLLFilePath
{
get { return _dllFilePath; }
set { _dllFilePath = value; OnPropertyChanged("DLLFilePath"); }
}

/// <summary>
/// A list of functions that resides within this DLL.
/// </summary>
public ObservableCollection<Function> DLLFunctions
{
get { return _dllFunctions; }
set { _dllFunctions = value; OnPropertyChanged("DLLFunctions"); }
}



#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
}
```

Instrument.cs

```
using Newtonsoft.Json;
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data
{
public enum MTSInstrumentType
{
RF_Instrument,
PXI,
```

```
PowerSupply
}

[Serializable]
public class Instrument : INotifyPropertyChanged
{
private string _instrumentName;
private MTSInstrumentType _instrumentType;
private int _instrumentID;
private string _serialNumber;

[JsonIgnore]
[XmlIgnore]
public string DisplayName => string.Format("{0} ({1})", this.InstrumentName, this.BoardNumber);
[JsonIgnore]
[XmlIgnore]
public int BoardNumber { get { return GetBoardNumberFromInstrumentName(); } } //Switch to get
set.

//Currently also stores the board number.
public string InstrumentName
{
get { return _instrumentName; }
set { _instrumentName = value; OnPropertyChanged("InstrumentName"); }
}

public MTSInstrumentType InstrumentType
{
get { return _instrumentType; }
set { _instrumentType = value; OnPropertyChanged("InstrumentType"); }
}
//public int InstrumentID
//{
//    get { return _instrumentID; }
//    set { _instrumentID = value; OnPropertyChanged("InstrumentID"); }
//}
public string SerialNumber
{
get { return _serialNumber; }
set { _serialNumber = value; OnPropertyChanged("SerialNumber"); }
}
```

```
#region Methods
public string SeparateBoardNumberFromInstrumentName()
{
if (!string.IsNullOrEmpty(this.InstrumentName)) //If it is not null or empty string, then proceed with regex processing.
{
return this.InstrumentName.Split(' ')[0];
}
else { return string.Empty; }
}
public int GetBoardNumberFromInstrumentName()
{
if (!string.IsNullOrEmpty(this.InstrumentName)) //If it is not null or empty string, then proceed with regex processing.
{
return int.Parse(System.Text.RegularExpressions.Regex.Match(this.InstrumentName.Split(' ')[1], @"\d+").Value);
}
else { return 0; }
}
[JsonIgnore]
[XmlIgnore]
public ObservableCollection<string> Ports
{
get { return BackendConstants.GetPorts(InstrumentName); }
}

//Out of use, can remove.
private ObservableCollection<string> GetPorts(string instrumentName)
{
var ports = new ObservableCollection<string>();
switch (instrumentName)
{
case "RF100":
ports = new ObservableCollection<string>(){"P1","P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9", "P10",
"P11", "P12", "P13", "P14","P15", "P16", "MA","MB","MC","MD"};
break;
case "RF110":
ports = new ObservableCollection<string>(){"P1","P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9", "P10",
"P11", "P12", "P13", "P14","P15", "P16", "HF_P1", "HF_P2", "HF_P3", "HF_P4"};
```

```
break;
case "RF200":
ports = new ObservableCollection<string>(){ "M1","M2", "M3", "M4", "M5", "M6", "M7", "M8", "M9",
"M10",
"M11", "M12", "M13", "M14", "M16", "M17","M18","M19","M20", "M21", "M22", "M23", "M24",
"M25", "M26" };
break;
case "RF210":
ports = new ObservableCollection<string>(){ "M1","M2", "M3", "M4", "M5", "M6", "M7", "M8", "M9",
"M10",
"M11", "M12", "M13", "M14", "M16", "HF_M1", "HF_M2", "HF_M3", "HF_M4", "HF_M5", "HF_M6",
"HF_M7", "HF_M8" };
break;
case "DPS":
ports = new ObservableCollection<string>(){ "LPS-1", "LPS-2", "LPS-3", "LPS-4", "LPS-5", "LPS-
6", "LPS-7", "LPS-8",
"HPC-1", "HPC-2", "HPC-3", "HPC-4",
"GPIO1","GPIO2","GPIO3","GPIO4","GPIO5","GPIO6","GPIO7","GPIO8","GPIO9","GPIO10","GPI
O11","GPIO12","GPIO13","GPIO14",
"GPIO15","GPIO16","GPIO17","GPIO18","GPIO19","GPIO20","GPIO21","GPIO22","GPIO23","GPI
O24","GPIO25","GPIO26","GPIO27",
"GPIO28","GPIO29","GPIO30","GPIO31","GPIO32",
"AI_14","AI_Sense","12V_Sw_Out","AI_4","AI_8","AI_24","AI_29","AI_31","PFI_1","+5V_Sw_Out",
"28VDC"

};
break;
}
return ports;
}
#endregion

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
```

}

MerlinProject.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models.CalModels;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.Data.Models.TestModels;
using MerlinTestStudio_Demo_Telerik.GraphViewModels;
using MerlinTestStudio_Demo_Telerik.UserControls;
using MerlinTestStudio_Demo_Telerik.UserControls.Calibration;
using MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager;
using MerlinTestStudio_Demo_Telerik.UserControls.Patterns;
using MerlinTestStudio_Demo_Telerik.UserControls.TestConditions;
using MerlinTestStudio_Demo_Telerik.UserControls.TestLimits;
using MerlinTestStudio_Demo_Telerik.UserControls.TestSequences;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.CalibrationViewModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using MT.TestStudio.Exceptions;
using MT.TestStudio.GUI.User_Controls;
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using System.Xml;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data.Models
{
public interface IContextMenuProvider
{
/// <summary>
/// Returns a new list of context menu items.
```

```
/// </summary>
List<ContextMenuItem> GetContextMenuItemsSource { get; }
}

///[Serializable]
///public class ProjectFile : INotifyPropertyChanged
///{
///    private string _filePath = string.Empty;
///
///    [XmlIgnore]
///    public string Name => Path.GetFileNameWithoutExtension(FilePath); //File Name with
extension.
///
///    [XmlIgnore]
///    public string FileName => Path.GetFileName(FilePath); //File Name w/ Extension.
///    [XmlAttribute]
///    public string FilePath
///    {
///        get { return _filePath; }
///        set { _filePath = value; OnPropertyChanged("FilePath"); OnPropertyChanged("FileName");
OnPropertyChanged("Name"); }
///    }
///    [XmlIgnore]
///    public Type DataType { get; private set; }
///    [XmlAttribute]
///    public string FileDataTypeName
///    {
///        get { return this.DataType == null ? string.Empty : this.DataType.FullName; }
///        set { this.DataType = value == null ? null : Type.GetType(value); }
///    }
///
///    #region Constructors
///    public ProjectFile() { } //De-serialization Constructor.
///    public ProjectFile(string filePath, Type dataType)
///    {
///        FilePath = filePath;
///        this.DataType = dataType;
///
///        //this.PVM = CreatePaneViewModelForFile();
///    }
///    #endregion
///
```

```
///    [XmlIgnore]
///    public ProjectFolder ParentFolder { get; set; }
///    //[XmlIgnore]
///    //public PaneViewModel PVM { get; set; }
///
///    //Method for creating PaneViewModels based on data from project files.
///    //public PaneViewModel CreatePaneViewModelForFile()
///    //{
///    //    Type dataViewType;
///    //
///    //    switch(DataType.GetType().Name)
///    //    {
///    //        case nameof(CalDataModel): dataViewType = typeof(CalibrationConfiguratorView);
break;
///    //        case nameof(PinMapModel): dataViewType = typeof(CalibrationConfiguratorView);
break;
///    //        default: dataViewType = null; break;
///    //    }
///    //
///    //    return new PaneViewModel(dataViewType, this, this.ParentFolder.ParentProject,
this.ParentFolder, this.ParentFolder.ParentProject.MVM) { Header = this.Name, IsDocument =
true };
///    //    //return new PaneViewModel(typeof(CalibrationConfiguratorView), newCdm,
folder.ParentProject, folder, this) { Header = uniqueFileName, IsDocument = true, ParentProject =
folder.ParentProject };
///    //}
///
///    #region INPC Members
///    [field: NonSerialized]
///    public event PropertyChangedEventHandler PropertyChanged;
///    private void OnPropertyChanged(string propertyName)
///    {
///        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
///    }
///
///    #endregion
///}

[Serializable]
[XmlType("ProjectFolder")]
public class ProjectFolder : INotifyPropertyChanged
{
```

```
private ObservableCollection<object> _folderItems = new ObservableCollection<object>();

[XmlAttribute]
public string FolderName { get; set; }
[XmlAttribute]
public ProjectFileSection Section { get; set; }
[XmlIgnore]
public MerlinProject ParentProject { get; set; }

//public ObservableCollection<object> SubFolders { get; set; }
[XmlIgnore]
public ObservableCollection<object> FolderItems
{
get { return _folderItems; }
set { _folderItems = value; OnPropertyChanged("FolderItems"); }
}

public List<string> Files { get; set; }

[XmlAttribute]
public bool IsExpanded { get; set; } //IsExpanded Project Tree Binding.

//Recursive save checker.
public void CheckSaved()
{
foreach (var obj in FolderItems) //Internalize this line with the save logic of each "data model" in the future
{
if (obj is ProjectFolder)
{
(obj as ProjectFolder).CheckSaved();
}
else if (obj is PVM)
{
(obj as PVM).IsSaveRequired = false;
}
}
}

//Recusively gets all the panes that have changes needing to be saved
public List<PVM> GetSaveRequiredPanes()
{
```

```
List<PVM> panes = new List<PVM>();

foreach (var obj in FolderItems)
{
if (obj is ProjectFolder)
{
panes.AddRange((obj as ProjectFolder).GetSaveRequiredPanes());
}
else if (obj is PVM)
{
if ((obj as PVM).IsSaveRequired == true) { panes.Add((obj as PVM)); }
}
}

return panes;
}

//Recusively gets pane of header parameter.
public PVM GetPaneFromHeader(string header)
{
PVM paneHeaderFound = null;

foreach (var obj in FolderItems)
{
if (obj is ProjectFolder)
{
var value = (obj as ProjectFolder).GetPaneFromHeader(header);
if(value != null)
{
paneHeaderFound = value;
}
}
else if (obj is PVM)
{
if ((obj as PVM).Header == header)
{
paneHeaderFound = obj as PVM;
}
}
}

return paneHeaderFound;
```

```
}
//TEMP
public PVM GetPaneFromHeaderNoExt(string header)
{
PVM paneHeaderFound = null;

foreach (var obj in FolderItems)
{
if (obj is ProjectFolder)
{
var value = (obj as ProjectFolder).GetPaneFromHeader(header);
if (value != null)
{
paneHeaderFound = value;
}
}
else if (obj is PVM)
{
if (Path.GetFileNameWithoutExtension((obj as PVM).Header) == header)
{
paneHeaderFound = obj as PVM;
}
}
}

return paneHeaderFound;
}

//Recusively gets all the panes.
public List<PVM> GetAllDocumentPanes()
{
List<PVM> panes = new List<PVM>();

foreach (var obj in FolderItems)
{
if (obj is ProjectFolder)
{
panes.AddRange((obj as ProjectFolder).GetAllDocumentPanes());
}
else if (obj is PVM)
{
panes.Add((obj as PVM));
```

```
}
}

return panes;
}

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}

[Serializable]
public class TestDataInfo //Only used for serializing the base project test data for import
comparison.
{
[XmlAttribute]
public int TestID { get; set; }
[XmlAttribute]
public int TestNumber { get; set; }
[XmlAttribute]
public string TestName { get; set; }
[XmlAttribute]
public string TestUnits { get; set; }
}

[Serializable]
[XmlRoot("MerlinProject")]
public class MerlinProject : INotifyPropertyChanged
{
#region Private Members
private string _projectName = string.Empty;
private string _projectDirectory= string.Empty;
private string _productName = string.Empty;
private string _testProgram = string.Empty;
private string _userTargetSystem = string.Empty;
```

```
private ObservableCollection<ProjectFolder> _projectTree = new
ObservableCollection<ProjectFolder>();

private ObservableCollection<ISequenceItem> _sequenceItems = new
ObservableCollection<ISequenceItem>(); //_allGroups
private ObservableCollection<Test> _tests = new ObservableCollection<Test>(); //_allTests
private ObservableCollection<Data.Instrument> _instruments = new
ObservableCollection<Data.Instrument>(); //_allInstruments
private ObservableCollection<Data.PinModel> _pins = new
ObservableCollection<Data.PinModel>(); //_allPins
private ObservableCollection<Data.DLL> _dlls = new ObservableCollection<Data.DLL>();
//_allDLLs
private ObservableCollection<Data.ParameterMapTag> _parameterMapTags = new
ObservableCollection<Data.ParameterMapTag>(); //_allParameterMapTags

private PinMapModel _pinMapDataModel = new PinMapModel();
//private ObservableCollection<IFileData> _testParametersDefinitions = new
ObservableCollection<IFileData>();
private ObservableCollection<IFileData> _testLimitsDefinitions = new
ObservableCollection<IFileData>();
private ObservableCollection<CalLimitsModel> _calibrationLimits = new
ObservableCollection<CalLimitsModel>();
private ObservableCollection<CalDataModel> _calibrationDefinitions = new
ObservableCollection<CalDataModel>();
private ObservableCollection<WaveformModel> _waveforms = new
ObservableCollection<WaveformModel>();
private ObservableCollection<IFileData> _digitalPatterns = new
ObservableCollection<IFileData>();
#endregion

#region Public Members
#region Project Info and Properties
public double Version { get; set; }
public string ProjectName { get { return _projectName; } set { _projectName = value;
OnPropertyChanged("ProjectName"); } }

[XmlIgnore]
public string ProjectFileName { get { return string.Format(@"{0}{1}", ProjectName,
BackendConstants.MerlinProjectExtension); } }

//Indicates where the project folder is located, not the project file.
public string ProjectDirectory { get { return _projectDirectory; } set { _projectDirectory = value;
```

```
OnPropertyChanged("ProjectDirectory"); } }

[XmlIgnore]
public string ProjectFilePath { get { return Path.Combine(ProjectDirectory, ProjectName,
ProjectFileName); } }
//public string ProjectNotes { get; set; }

public string ProductName { get { return _productName; } set { _productName = value;
OnPropertyChanged("ProductName"); } }

public string TestProgram { get { return _testProgram; } set { _testProgram = value;
OnPropertyChanged("TestProgram"); } }

public string UserTargetSystem
{
get { return _userTargetSystem; }
set
{
_userTargetSystem = value;
OnPropertyChanged("UserTargetSystem");

foreach (CalDataModel cdm in this.CalibrationDefinitions)
{
cdm.GlobalSystemVarient = value;
}
//foreach(var obj in GetProjectFolder(ProjectFileSection.Calibration_Definitions).FolderItems)
//{
//    (obj as PaneViewModel).IsSaveRequired = true;
//}
}
}

[XmlIgnore]
public string PinMapFilePath //DefaultPinMapFilePath
{
get { return Path.Combine(this.ProjectDirectory, this.ProjectName,
BackendConstants.ConnectionManagerDirString, string.Format("{0} PinMap{1}",
this.ProjectName, BackendConstants.TestStudioPinMapExtension)); } set { } //Do Nothing.
}
[XmlIgnore]
public string BaseTestInfoFilePath
{
```

```
get { return Path.Combine(this.GetProjectFileSectionDirectory(ProjectFileSection.Project_Data),
"ProjectBaseTestInfo.xml"); } set { } //Do Nothing.
}

#region Temp File Paths for Sequence and Parameters data
[XmlIgnore]
public string TestParametersFilePath
{
get { return Path.Combine(this.ProjectDirectory, this.ProjectName,
BackendConstants.TestParametersDirString, "Test_Parameters_Data.xml"); } set { } //Do Nothing.
}
[XmlIgnore]
public string ParameterMapTagsFilePath
{
get { return Path.Combine(this.ProjectDirectory, this.ProjectName,
BackendConstants.TestParametersDirString, "Parameter_Mapping_Data.xml"); } set { } //Do
Nothing.
}
[XmlIgnore]
public string DllReferencesFilePath
{
get { return Path.Combine(this.ProjectDirectory, this.ProjectName, BackendConstants.DllDirString,
"Dll_References_Data.xml"); } set { } //Do Nothing.
}
[XmlIgnore]
public string TestSequencesFilePath
{
get { return Path.Combine(this.ProjectDirectory, this.ProjectName,
BackendConstants.TestSequencesDirString, "DUT_Test_Sequence_Data.xml"); } set { } //Do
Nothing.
}
#endregion

#endregion

#region Internal Data
//public string SequenceDataModelFilePath { get; set; }
//[XmlIgnore]
//public SequenceDataModel SequenceDataObject
//{
//    get;
//    set;
```

```
//}

[XmlIgnore]//_allGroups
public ObservableCollection<ISequenceItem> SequenceItems //Interface of objetcs is needed to ensure GUI binding.
{
get { return _sequenceItems; }
set { _sequenceItems = value; OnPropertyChanged("SequenceItems"); }
}

[XmlIgnore]//_allTests
public ObservableCollection<Test> Tests //The Main Merlin Project backend Test Collection for Test Synchronization.
{
get { return _tests; }
set { _tests = value; OnPropertyChanged("Tests"); }
}

[XmlIgnore]
public PinMapModel PinMapDataModel
{
get { return _pinMapDataModel;  } set { _pinMapDataModel = value; OnPropertyChanged("PinMapDataModel"); }
}

[XmlIgnore]//_allDLLs
public ObservableCollection<Data.DLL> DLLs
{
get { return _dlls; }
set { _dlls = value; OnPropertyChanged("DLLs"); }
}
[XmlIgnore]//_allParameterMapTags
public ObservableCollection<Data.ParameterMapTag> ParameterMapTags
{
get { return _parameterMapTags; }
set { _parameterMapTags = value; OnPropertyChanged("ParameterMapTags"); }
}
#endregion

[XmlAttribute]
public string IsExpanded { get; set; } //IsExpanded Project Tree Binding.
```

```
public ObservableCollection<ProjectFolder> ProjectTree
{
get { return _projectTree; }
set { _projectTree = value; OnPropertyChanged("ProjectTree"); }
}

#region Collection File References

private ObservableCollection<object> _testLimits = new ObservableCollection<object>(); //TEST
[XmlIgnore] //TEST
public ObservableCollection<object> TestLimits //TEST
{
get { return _testLimits; }
set { _testLimits = value; OnPropertyChanged("TestLimits"); }
}

public List<string> CalibrationLimitsFileReferences { get; set; }
[XmlIgnore]
public ObservableCollection<CalLimitsModel> CalibrationLimits { get { return _calibrationLimits; }
set { _calibrationLimits = value; OnPropertyChanged("CalibrationLimits"); } }

public List<string> CalibrationDefinitionFileReferences { get; set; }
[XmlIgnore]
public ObservableCollection<CalDataModel> CalibrationDefinitions { get { return
_calibrationDefinitions; } set { _calibrationDefinitions = value;
OnPropertyChanged("CalibrationDefinitions"); }}

public List<string> WaveformFileReferences { get; set; } //Outdated.
[XmlIgnore]//Outdated.
public ObservableCollection<WaveformModel> Waveforms //Outdated.
{
get { return _waveforms; } set { _waveforms = value; OnPropertyChanged("Waveforms"); }
}

//TEMP LINK
[XmlIgnore]
public List<PVM> AvailableWaveforms
{
get
{
var folder = this.GetProjectFolder(ProjectFileSection.Waveforms);
```

```
var availableWaveforms = folder.GetAllDocumentPanes();

return availableWaveforms;
}
}
//TEMP LINK
public void WaveformsChanged()
{
OnPropertyChanged("AvailableWaveforms");
}

//TEMP LINK
[XmlIgnore]
public List<DigiTimeObj> AvailableTimeSets //Gets time sets from every digital timing model in the project.
{
get
{
List<DigiTimeObj> rVal = new List<DigiTimeObj>();
if (this.DigitalPatterns != null)
{
foreach (var model in this.DigitalPatterns.Where(model => model is DigitalTimingModel))
{
rVal.AddRange((model as DigitalTimingModel).GetTimeSets()); //NOTE: No name check means time sets from different files can have the same name leading to duplicates.
}
}
return rVal;
}
}
//TEMP LINK
public void TimeSetsChanged()
{
OnPropertyChanged("AvailableTimeSets");
}

public List<string> DigitalPatternFileReferences { get; set; }
[XmlIgnore]
public ObservableCollection<IFileData> DigitalPatterns
{
get { return _digitalPatterns; }
```

```
set { _digitalPatterns = value; OnPropertyChanged("DigitalPatterns"); }
}

#endregion

private List<TestDataInfo> _baseProjectTestInfo = new List<TestDataInfo>();
[XmlIgnore]
public List<TestDataInfo> BaseProjectTestInfo
{
get { return _baseProjectTestInfo; }
set
{
if(value != null)
{
_baseProjectTestInfo = value;
}
}
}
//Base test info data for accurate comparison against imported test files.
//ON IMPORT: Construct all the base data tests => Compare all base test info data against data imports => Once all match, Import based on TestID using FirstOrDefault();
//FAST IDEA: Maybe dont even worry about importing parameters, worry about Test Sync first within TEST LIMITS...

[XmlIgnore]
public MainWindowViewModel MVM => DataStorage.MainViewModel;

private MerlinSystem _projectSystemCache;
public MerlinSystem ProjectSystemCache
{
get { return _projectSystemCache; }
set { _projectSystemCache = value; OnPropertyChanged("MySystem"); }
}
#endregion

#region Constructor
public MerlinProject() { } //De-Serialization Constructor.
public MerlinProject(string FileName)
{
this.Version = BackendConstants.CurrentMerlinProjectVersion;
this.ProjectName = Path.GetFileNameWithoutExtension(FileName);
this.ProjectDirectory = Path.GetDirectoryName(FileName);
```

```
//this.MVM = mvm;

var defaultTestParamTags = new List<Data.ParameterMapTag>()
{
new Data.ParameterMapTag() { AttributeType = Data.TestAttributeType.Test_Number,
ColumnIndex = 0, ColumnName = "Test Number", IsAutoGen = false},
new Data.ParameterMapTag() { AttributeType = Data.TestAttributeType.Test_Name,
ColumnIndex = 1, ColumnName = "Test Name", IsAutoGen = false },
new Data.ParameterMapTag() { AttributeType = Data.TestAttributeType.Test_Unit, ColumnIndex
= 2, ColumnName = "Unit", IsAutoGen = false },
new Data.ParameterMapTag() { AttributeType = Data.TestAttributeType.Function_Call,
ColumnIndex = 3, ColumnName = "Function Call", IsAutoGen = false },
new Data.ParameterMapTag() { AttributeType = Data.TestAttributeType.Group, ColumnIndex = 4,
ColumnName = "Group", IsAutoGen = false }
};
foreach (Data.ParameterMapTag pmt in defaultTestParamTags) //Add the default parameter map
tags
{
this.ParameterMapTags.Add(pmt);
}

this.PinMapDataModel.FilePath = this.PinMapFilePath; //Set the Default PinMapFilePath for
project creation

//Temp build of default project tree, needs to be made dynamic.
foreach (KeyValuePair<ProjectFileSection, string> kvp in
BackendConstants.ProjectFileSectionKvPs)
{
ProjectFolder fileSection = new ProjectFolder() { FolderName = kvp.Value, Section = kvp.Key,
ParentProject = this };

switch (kvp.Key)
{
case ProjectFileSection.Product_Configurations:
//fileSection.FolderItems.Add(new ProductConfigViewModel(typeof(ProductConfiguration), this) {
Header = "Production Config", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder =
fileSection, Icon = "Settingsd.ico" });
//fileSection.FolderItems.Add(new ProductConfigViewModel(typeof(ProductConfiguration), this) {
Header = "Development Config", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder
= fileSection, Icon = "Settingsd.ico" });
break;
case ProjectFileSection.Project_Data:
```

```
continue; //Do not add the folder, continue the loop without executing the addition function.
case ProjectFileSection.Test_Limits:
break;
case ProjectFileSection.Test_Parameters:
fileSection.FolderItems.Add(new ConditionsViewModel(typeof(ConditionsData),
this.TestParametersFilePath, this, fileSection) { Header = "Test Parameters", IsDocument = true,
IsStatic = true, IsHidden = true, ParentFolder = fileSection });
break;
case ProjectFileSection.Test_Sequences:
//fileSection.FolderItems.Add(new PaneViewModel(typeof(Session_Load), this) { Header =
"Session Load", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder = fileSection });
//fileSection.FolderItems.Add(new PaneViewModel(typeof(DUT_Start_Test), this) { Header = "DUT
Start Test", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder = fileSection });
fileSection.FolderItems.Add(new DUT_TestViewModel(typeof(DUT_Test),
this.TestSequencesFilePath, this, fileSection) { Header = "DUT Test", IsDocument = true, IsStatic
= true, IsHidden = true, ParentFolder = fileSection, Icon = "TaskList.png" });
//fileSection.FolderItems.Add(new PaneViewModel(typeof(DUT_End_Test), this) { Header = "DUT
End Test", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder = fileSection });
//fileSection.FolderItems.Add(new PaneViewModel(typeof(SessionUnload), this) { Header =
"Session Unload", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder = fileSection
});
break;
case ProjectFileSection.Calibration_Definitions:
break;
case ProjectFileSection.Calibration_Limits:
break;
case ProjectFileSection.Connection_Manager:
fileSection.FolderItems.Add(new PinMapViewModel(typeof(PinMap), this.PinMapDataModel, this,
fileSection) { Header = string.Format("{0} Pin Map", this.ProjectName), IsDocument = true, IsStatic
= true, IsHidden = true, Icon = "Diagram.png" });
break;
case ProjectFileSection.Connection_Diagrams:
break;
case ProjectFileSection.Waveforms:
break;
case ProjectFileSection.Digital_Patterns:
break;
case ProjectFileSection.Dll:
break;
}
ProjectTree.Add(fileSection);
}
```

```
}
#endregion

#region Methods

public ProjectFolder GetProjectFolder(ProjectFileSection fileSection)
{
return this.ProjectTree.Where(x => x.Section == fileSection).FirstOrDefault();
}
public string GetSubDirectory(ProjectFileSection fileSection)
{
string subDir = string.Empty;
switch (fileSection)
{
case ProjectFileSection.Product_Configurations: subDir = Path.Combine(ProjectDirectory,
ProjectName, BackendConstants.ProductConfigurationsDirString); break;
case ProjectFileSection.Project_Data: subDir = Path.Combine(ProjectDirectory, ProjectName,
BackendConstants.ProjectDataDirString); break;
case ProjectFileSection.Test_Limits: subDir = Path.Combine(ProjectDirectory, ProjectName,
BackendConstants.TestLimitsDirString); break;
case ProjectFileSection.Test_Parameters: subDir = Path.Combine(ProjectDirectory, ProjectName,
BackendConstants.TestParametersDirString); break;
case ProjectFileSection.Test_Sequences: subDir = Path.Combine(ProjectDirectory, ProjectName,
BackendConstants.TestSequencesDirString); break;
case ProjectFileSection.Calibration_Definitions: subDir = Path.Combine(ProjectDirectory,
ProjectName, BackendConstants.CalibrationDefinitionsDirString); break;
case ProjectFileSection.Calibration_Limits: subDir = Path.Combine(ProjectDirectory,
ProjectName, BackendConstants.CalibrationLimitsDirString); break;
case ProjectFileSection.Connection_Manager: subDir = Path.Combine(ProjectDirectory,
ProjectName, BackendConstants.ConnectionManagerDirString); break;
case ProjectFileSection.Connection_Diagrams: subDir = Path.Combine(ProjectDirectory,
ProjectName, BackendConstants.ConnectionDiagramsDirString); break;
case ProjectFileSection.Waveforms: subDir = Path.Combine(ProjectDirectory, ProjectName,
BackendConstants.WaveformsDirString); break;
case ProjectFileSection.Digital_Patterns: subDir = Path.Combine(ProjectDirectory, ProjectName,
BackendConstants.DigitalPatternsDirString); break;
case ProjectFileSection.Dll: subDir = Path.Combine(ProjectDirectory, ProjectName,
BackendConstants.DllDirString); break;
}
return subDir;
}
```

```
//Provides the sub-directory from the given ProjectFileSection enum key. (Needs to be tested)
public string GetProjectFileSectionDirectory(ProjectFileSection fileSection)
{
string enumAssociation = string.Empty;
foreach (var kvp in BackendConstants.ProjectFileSectionKvPs.Where(kvp => kvp.Key ==
fileSection)) { enumAssociation = kvp.Value; }

return Path.Combine(this.ProjectDirectory, this.ProjectName, enumAssociation);
}

//Creates the project directory structure.
public void CreateProjectFileStructure()
{
DirectoryInfo dataDir = Directory.CreateDirectory(Path.Combine(this.ProjectDirectory,
this.ProjectName));

foreach(KeyValuePair<ProjectFileSection, string> kvp in
BackendConstants.ProjectFileSectionKvPs)
{
dataDir.CreateSubdirectory(kvp.Value);
if(kvp.Key == ProjectFileSection.Project_Data)
{
DirectoryInfo pd_dir = dataDir.CreateSubdirectory(kvp.Value);
pd_dir.CreateSubdirectory("Cal Data");
}
else if (kvp.Key == ProjectFileSection.Waveforms)
{
//DirectoryInfo w_dir = dataDir.CreateSubdirectory(kvp.Value);
//foreach(string subd in BackendConstants.WaveformSections) { w_dir.CreateSubdirectory(subd);
}
}
else
{
dataDir.CreateSubdirectory(kvp.Value);
}
}

this.SaveProjectFile(ProjectFilePath);

//Find a better spot to place into
#region default data / file saving on project creation.
//var TestParameterService = new ServiceLocator().testParameterService;
```

```
//var defaultTestParamTags = new List<Data.ParameterMapTag>()
//                {
//                    new Data.ParameterMapTag() { AttributeType =
Data.TestAttributeType.Test_Number,  ColumnIndex = 0, ColumnName = "Test Number",
IsAutoGen = false},
//                    new Data.ParameterMapTag() { AttributeType =
Data.TestAttributeType.Test_Name, ColumnIndex = 1, ColumnName = "Test Name", IsAutoGen =
false },
//                    new Data.ParameterMapTag() { AttributeType =
Data.TestAttributeType.Test_Unit, ColumnIndex = 2, ColumnName = "Unit", IsAutoGen = false },
//                    new Data.ParameterMapTag() { AttributeType =
Data.TestAttributeType.Function_Call, ColumnIndex = 3, ColumnName = "Function Call",
IsAutoGen = false },
//                    new Data.ParameterMapTag() { AttributeType = Data.TestAttributeType.Group,
ColumnIndex = 4, ColumnName = "Group", IsAutoGen = false }
//                };
//foreach (Data.ParameterMapTag pmt in defaultTestParamTags)
//    TestParameterService.AddColumn(pmt.ColumnName, typeof(string), pmt);
//
//var bf = new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
//DataManagement.SaveParameterMapTagData(ref bf, Path.Combine(this.ProjectDirectory,
this.ProjectName, "Project Files"));
//
//var bf2 = new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
//DataManagement.SaveInstrumentsData(ref bf, Path.Combine(this.ProjectDirectory,
this.ProjectName, "Project Files"));
#endregion

}

#endregion

#region Hint Path Creation and Loading
public string ProjectHintPathCreator(string FullPath)
{
string hintPath = string.Empty;
string hintPathDir = Path.Combine(this.ProjectDirectory, this.ProjectName);

string[] hpDirArr = hintPathDir.Split(Path.DirectorySeparatorChar);
List<string> fpDirArr = FullPath.Split(Path.DirectorySeparatorChar).ToList();

for (int i = 0; i < hpDirArr.Count(); i++)
```

```
{
fpDirArr.Remove(hpDirArr[i]);
}

hintPath = Path.Combine(fpDirArr.ToArray());

return hintPath;
}
public static string ProjectFullPathCreator(MerlinProject project, string HintPath)
{
string fullPath = Path.Combine(project.ProjectDirectory, project.ProjectName, HintPath);
return fullPath;
}

#region GPT Hint Path method recommendations
//public string ProjectHintPathCreator(string FullPath)
//{
//    // Make sure FullPath is not null or empty
//    if (string.IsNullOrEmpty(FullPath))
//    {
//        throw new ArgumentException("FullPath cannot be null or empty.");
//    }
//
//    string hintPath = string.Empty;
//    string hintPathDir = Path.Combine(this.ProjectDirectory, this.ProjectName);
//
//    // Split hintPathDir and FullPath into arrays
//    string[] hpDirArr = hintPathDir.Split(Path.DirectorySeparatorChar);
//    string[] fpDirArr = FullPath.Split(Path.DirectorySeparatorChar);
//
//    // Use Except to remove the common elements from fpDirArr
//    string[] result = fpDirArr.Except(hpDirArr).ToArray();
//
//    // Combine the remaining elements of result into a single string
//    hintPath = Path.Combine(result);
//
//    return hintPath;
//}

//public static string ProjectFullPathCreator(MerlinProject project, string HintPath)
//{
//    // Make sure project and HintPath are not null or empty
```

```
//    if (project == null)
//    {
//        throw new ArgumentException("project cannot be null.");
//    }
//    if (string.IsNullOrEmpty(HintPath))
//    {
//        throw new ArgumentException("HintPath cannot be null or empty.");
//    }
//
//    // Make sure ProjectDirectory and ProjectName are not null or empty
//    if (string.IsNullOrEmpty(project.ProjectDirectory))
//    {
//        throw new ArgumentException("ProjectDirectory cannot be null or empty.");
//    }
//    if (string.IsNullOrEmpty(project.ProjectName))
//    {
//        throw new ArgumentException("ProjectName cannot be null or empty.");
//    }
//
//    // Combine the ProjectDirectory, ProjectName, and HintPath into a full file path
//    string fullPath = Path.Combine(project.ProjectDirectory, project.ProjectName, HintPath);
//    return fullPath;
//}
#endregion

#endregion

public List<PVM> GetAllProjectDocumentPanes()
{
List<PVM> panes = new List<PVM>();

foreach (ProjectFolder folder in this.ProjectTree)
{
panes.AddRange(folder.GetAllDocumentPanes());
}

return panes;
}

//Only tests against TestID, TestNumber, TestName, and Units.
//To Use On Load for any project file containing test data needing to be synced.
public bool MatchCurrentTestInfo(IEnumerable<Test> testsToMatch)
```

```
{
this.BaseProjectTestInfo = GetCurrentTestInfo(); //Update to match with latest test data.

//Check test counts first.
if (BaseProjectTestInfo.Count() != testsToMatch.Count())
{
return false; //Automatically indicates there isn't the same amount of tests.
}

//Then check if info data matches the input tests.
for (int i = 0; i < BaseProjectTestInfo.Count(); i++)
{
TestDataInfo baseTest = BaseProjectTestInfo[i];
Test inputTest = testsToMatch.ElementAt(i);
if( baseTest.TestNumber != inputTest.TestNumber || baseTest.TestName != inputTest.TestName
|| baseTest.TestUnits != inputTest.Units) //Test ID not in use.
{
return false;
}
}

//If all checks pass, assume tests match.
return true;
}

//Returns the Base Test Data Info for this project
private List<TestDataInfo> GetCurrentTestInfo()
{
List<TestDataInfo> testInfo = new List<TestDataInfo>();

foreach (Test test in this.Tests) //Only grabs from 'Tests', where the only set for field is old .bin
serialization :/
{
TestDataInfo infoToAdd = new TestDataInfo()
{
TestID = test.TestID,
TestNumber = test.TestNumber,
TestName = test.TestName,
TestUnits = test.Units
};
testInfo.Add(infoToAdd);
}
```

```
return testInfo;
}


public void SaveProjectFile(string FileName)
{
try
{
//Upon save, update version to match current class version serialization.
this.Version = BackendConstants.CurrentMerlinProjectVersion;

//Create references before saving. (OLD)
//CalibrationDefinitionFileReferences = (CalibrationDefinitions.Select(mp =>
this.ProjectHintPathCreator(mp.FilePath))).ToList();
//CalibrationLimitsFileReferences = (CalibrationLimits.Select(mp =>
this.ProjectHintPathCreator(mp.FilePath))).ToList();
//WaveformFileReferences = (Waveforms.Select(mp =>
this.ProjectHintPathCreator(mp.FilePath))).ToList(); // No longer needed.
//DigitalPatternFileReferences = (DigitalPatterns.Select(mp =>
this.ProjectHintPathCreator(mp.FilePath))).ToList();  // No longer needed.

//Creates all the file references needed under each parent folder. (NEW)
foreach (var pf in this.ProjectTree) //Should retreive sub directories too.
{
pf.Files = pf.GetAllDocumentPanes().Select(x => (string.IsNullOrEmpty(x.DataFilePath) ? "" :
this.ProjectHintPathCreator(x.DataFilePath))).ToList();
}

///Prev. used = FileMode.OpenOrCreate == attempt to fix extra characters saving.
using (FileStream stream = new FileStream(FileName, FileMode.Create, FileAccess.ReadWrite))
{
var xmlSerializer = new XmlSerializer(typeof(MerlinProject));
xmlSerializer.Serialize(stream, this);
}

//this.BaseProjectTestInfo = GetCurrentTestInfo();
using (FileStream stream = new FileStream(this.BaseTestInfoFilePath, FileMode.Create,
FileAccess.ReadWrite))
{
XmlSerializer xmlSerializer = new XmlSerializer(typeof(List<TestDataInfo>));
xmlSerializer.Serialize(stream, this.BaseProjectTestInfo);
```

```
}
}
catch (Exception ex)
{
MessageBox.Show(string.Format("An error occured during project save: {0} ({1}) \nError: {2}",
this.ProjectName, this.ProjectFilePath, ex.Message));
}
finally { Console.WriteLine(string.Format("Project Saved: {0} ({1})", this.ProjectName,
this.ProjectFilePath)); }
}

//public void SaveToXML(string FileName)
//{
//    try
//    {
//        //Upon save, update version to match current class version serialization.
//        this.Version = BackendConstants.CurrentMerlinProjectVersion;
//
//        this.SaveProjectFile(FileName);
//
//        /////Prev. used = FileMode.OpenOrCreate == attempt to fix extra characters saving.
//        //using (FileStream stream = new FileStream(FileName, FileMode.Create,
FileAccess.ReadWrite))
//        //{
//        //    var xmlSerializer = new XmlSerializer(typeof(MerlinProject));
//        //    xmlSerializer.Serialize(stream, this);
//        //}
//
//        #region Save Internal Data (Parameters, Sequence, Etc)
//        ///SaveTestItemsData(this.TestParametersFilePath);
//        ///SaveParameterMapTagData(this.ParameterMapTagsFilePath);
//        ///SaveDLLsData(this.DllReferencesFilePath);
//        ///SaveSequenceItemsData(this.TestSequencesFilePath);
//        #endregion
//    }
//    catch (Exception ex)
//    {
//        MessageBox.Show(string.Format("An error occured during project save: {0} ({1}) \nError:
{2}", this.ProjectName, this.ProjectFilePath, ex.Message));
//    }
//    finally { Console.WriteLine(string.Format("Project Saved: {0} ({1})", this.ProjectName,
this.ProjectFilePath)); }
```

```
//
//}

public static MerlinProject LoadFromXML(string FileName, out List<string> MissingFiles)
{
try
{
#region Deserialize the MTPROJ project file
MerlinProject projectLoaded;
var xmlSerializer = new XmlSerializer(typeof(MerlinProject));
using (FileStream stream = new FileStream(FileName, FileMode.Open))
{
projectLoaded = (MerlinProject)xmlSerializer.Deserialize(stream);
}
//projectLoaded.MVM = mvm;
#endregion Deserialize the MTPROJ project file

///Change directory to the loaded from directory.
string dirWithName = Path.GetDirectoryName(FileName); //Dir w/ project name folder
projectLoaded.ProjectDirectory = Directory.GetParent(dirWithName).FullName; //Project folder
directory. Set where ever this project is loaded from.
///Missing file path list
List<string> MissingProjectFilePaths = new List<string>();

#region Project version tolerant on load adjustments
//Version tolerent project file adjustments on load.
for (double i = projectLoaded.Version; i < BackendConstants.CurrentMerlinProjectVersion; i++)
//Increaments through all the past projhect versions so all fixes get applied.
{
switch (projectLoaded.Version)
{
case 1:
case 2:
break;
case 3: //Added hint paths so projects can be loaded outside the C:/MerlinTest/Projects/ directory.
projectLoaded.CalibrationDefinitionFileReferences =
(projectLoaded.CalibrationDefinitionFileReferences.Select(str =>
projectLoaded.ProjectHintPathCreator(str))).ToList();
projectLoaded.WaveformFileReferences = (projectLoaded.WaveformFileReferences.Select(str =>
projectLoaded.ProjectHintPathCreator(str))).ToList();
projectLoaded.DigitalPatternFileReferences =
(projectLoaded.DigitalPatternFileReferences.Select(str =>
```

```
projectLoaded.ProjectHintPathCreator(str))).ToList();
break;
case 4:///Create the Calibration Limits project folder that did not exist for prior versions of v4. Now moves to v5.
int CalLimitsIndex = 4;
KeyValuePair<ProjectFileSection, string> CalLimitsKvP = BackendConstants.ProjectFileSectionKvPs[CalLimitsIndex];
//Create the Physical Drive Directory.
DirectoryInfo dataDir = Directory.CreateDirectory(Path.Combine(projectLoaded.ProjectDirectory, projectLoaded.ProjectName));
dataDir.CreateSubdirectory(CalLimitsKvP.Value);
//Then create and insert the Folder Obj.
ProjectFolder fileSection = new ProjectFolder() { FolderName = CalLimitsKvP.Value, Section = CalLimitsKvP.Key, ParentProject = projectLoaded };
projectLoaded.ProjectTree.Insert(CalLimitsIndex, fileSection);
break;
case 5:
///WHEN THIS CODE EXECUTES: It's implied that only Test Limit files are being migrated and new project folder sections for those files are being added.
ProjectFolder projDataFolder = projectLoaded.ProjectTree.Where(x => x.Section == ProjectFileSection.Project_Data).FirstOrDefault();
if(projDataFolder != null) { projectLoaded.ProjectTree.Remove(projDataFolder); } //If folder found => Remove it from project tree.

KeyValuePair<ProjectFileSection, string> TestLimitsKvP = BackendConstants.ProjectFileSectionKvPs[2];
KeyValuePair<ProjectFileSection, string> TestParamsKvP = BackendConstants.ProjectFileSectionKvPs[3];
//Create the Physical Drive Directory.
DirectoryInfo projectDir = Directory.CreateDirectory(Path.Combine(projectLoaded.ProjectDirectory, projectLoaded.ProjectName));
projectDir.CreateSubdirectory(TestLimitsKvP.Value);
projectDir.CreateSubdirectory(TestParamsKvP.Value);

//Then create and insert the Folder Obj.
ProjectFolder limitsSection = new ProjectFolder() { FolderName = TestLimitsKvP.Value, Section = TestLimitsKvP.Key, ParentProject = projectLoaded };
projectLoaded.ProjectTree.Insert(1, limitsSection);
ProjectFolder paramsSection = new ProjectFolder() { FolderName = TestParamsKvP.Value, Section = TestParamsKvP.Key, ParentProject = projectLoaded };
projectLoaded.ProjectTree.Insert(2, paramsSection);
```

```
///Commented code below does not work since save SaveProjectFile() used the loaded data models to create all hint path references.
//projectLoaded.Version++; //increment by one to the correct project version for project file save.
//projectLoaded.SaveProjectFile(projectLoaded.ProjectFilePath); //Save the new new Test Limit File Hint References
//projectLoaded.Version--; //Decrement by one to keep the version based adjustments occuring in order if necessary.
MessageBox.Show("Upgrading the project version to v6... Please click 'Save All' once the project has loaded in order to save migrated file references.");
break;
case 6:
#region Convert JSON Cal Def to XML and change the default Cal Def extension
List<CalDataModel> jsonLoadedModels = new List<CalDataModel>();
ProjectFolder calDefsFolder = projectLoaded.ProjectTree.Where(x => x.Section == ProjectFileSection.Calibration_Definitions).FirstOrDefault();
if(calDefsFolder != null)
{
//Load all the JSON models.
foreach (string hintFileRef in (calDefsFolder.Files.Count() > 0 ? calDefsFolder.Files : projectLoaded.CalibrationDefinitionFileReferences))
{
string fileRef = ProjectFullPathCreator(projectLoaded, hintFileRef);
if (File.Exists(fileRef))
{
CalDataModel cdmLoaded = CalDataModel.LoadFromJson(fileRef);
if (cdmLoaded != null)
{
cdmLoaded.FilePath = fileRef; //Ensures the path is correct for later.
jsonLoadedModels.Add(cdmLoaded);
}
else { Console.WriteLine($"Project upgrade to v7, {fileRef} loaded null for Calibration Definition JSON to XML conversion with new extension."); }
}
else { Console.WriteLine($"Project upgrade to v7, Could not find {fileRef} for Calibration Definition JSON to XML conversion with new extension."); }
}
//Clear the old hint file refs (TEST)
calDefsFolder.Files.Clear();
//Convert all loaded models into XML with and save file with the new extension.
foreach (var model in jsonLoadedModels)
{
```

```
string oldFilePath = model.FilePath; //Store the old file path for deletion after XML save confirmation.
string newExtensionFilePath = Path.ChangeExtension(model.FilePath, BackendConstants.MTS_Cal_Config_Extension);

model.FilePath = newExtensionFilePath; //Save with the correct extension.
model.SaveToXml(newExtensionFilePath); //Saves to xml with the new extension.

File.Delete(oldFilePath); //Delete the old file that contains the old extension.

//calDefsFolder.Files.Remove(oldFilePath); //Remove old file path.
string newExtHintPathToAdd = projectLoaded.ProjectHintPathCreator(newExtensionFilePath);
//create the hint path so it's reloaded correctly later on PVM creation
calDefsFolder.Files.Add(newExtHintPathToAdd); //Add the new file path as a hint path.
}
//Remind user to click Save All when the conversion is done.
MessageBox.Show("Upgrading the project version to v7... Please click 'Save All' once the project has loaded in order to save migrated file references.");
}
else { Console.WriteLine($"Project upgrade to v7, Could not find Calibration Definitions folder (null) for JSON to XML conversion with new extension.") ; }
#endregion
break;
case 7:  //Next project version.
break;
}
projectLoaded.Version++; //Increment the version number of the project so all changes occur in order
}
#endregion Project version tolerant on load adjustments

#region Load Base Test Info File
try
{
if (File.Exists(projectLoaded.BaseTestInfoFilePath))
{
XmlSerializer XmlSerializer = new XmlSerializer(typeof(List<TestDataInfo>));
using (StreamReader FileStream = new StreamReader(projectLoaded.BaseTestInfoFilePath))
{
projectLoaded.BaseProjectTestInfo = (List<TestDataInfo>)XmlSerializer.Deserialize(FileStream);
}
}
```

```
else
{
Console.WriteLine("Project loaded no base test data...");
//MissingProjectFilePaths.Add(projectLoaded.BaseTestInfoFilePath);
}
}
catch (InvalidOperationException xmlEx) { MessageBox.Show(string.Format("Error occured
loading base test info XML.\n{0}\n{1}", xmlEx.Message, projectLoaded.BaseTestInfoFilePath)); }
catch (Exception regEx) { MessageBox.Show(regEx.Message); }
#endregion Load Base Test Info File
#region Load Pin Map
try
{
if (File.Exists(projectLoaded.PinMapFilePath))
{
projectLoaded.PinMapDataModel =
PinMapModel.ImportFromXml(projectLoaded.PinMapFilePath);
}
else { MissingProjectFilePaths.Add(projectLoaded.PinMapFilePath); }
#region 3.0.0.17 pin map loading code
//string xmlDataPath = Path.Combine(projectLoaded.ProjectDirectory,
projectLoaded.ProjectName, "Pin Map", string.Format("{0} PinMap.xml",
projectLoaded.ProjectName));
//if (File.Exists(xmlDataPath)) //.xml file
//{
//    XmlSerializer pinsXmlSerializer = new XmlSerializer(typeof(ObservableCollection<Data.Pin>));
//    using (FileStream pinsFileStream = new FileStream(xmlDataPath, FileMode.Open))
//    {
//        projectLoaded.Pins =
(ObservableCollection<Data.Pin>)pinsXmlSerializer.Deserialize(pinsFileStream);
//    }
//}
//else
//{
//    string newDataPath = Path.Combine(projectLoaded.ProjectDirectory,
projectLoaded.ProjectName, "Pin Map", string.Format("{0} PinMap{1}",
projectLoaded.ProjectName, BackendConstants.TestStudioPinMapExtension));
//    if (File.Exists(newDataPath))
//    {
//        projectLoaded.PinMapDataModel = PinMapModel.ImportPinMapXML(newDataPath);
//        projectLoaded.Pins = projectLoaded.PinMapDataModel.RfPins;
//        projectLoaded.DigitalPins = projectLoaded.PinMapDataModel.DigitalPins;
```

```
//        projectLoaded.PowerSupplyPins = projectLoaded.PinMapDataModel.PowerSupplyPins;
//    }
//}
#endregion
#region Temp. Pin Mapping Instrument Name Changing (5/9/2022)(Remove at a later date)
//For older projects convert the names of the old instruments so that the pin mapping instrument names aren't blank.
foreach (var (m, beforeChange) in from PinModel p in projectLoaded.PinMapDataModel.RfPins
from MappingModel m in p.mappings
where BackendConstants.ExistingInstruments.Contains(m.InstrumentName)
let beforeChange = m.InstrumentName
select (m, beforeChange))
{ m.InstrumentName = string.Format("{0} (1)", beforeChange); }
foreach (var (m, beforeChange) in from PinModel p in
projectLoaded.PinMapDataModel.DigitalPins
from MappingModel m in p.mappings
where BackendConstants.ExistingInstruments.Contains(m.InstrumentName)
let beforeChange = m.InstrumentName
select (m, beforeChange))
{ m.InstrumentName = string.Format("{0} (1)", beforeChange); }
foreach (var (m, beforeChange) in from PinModel p in
projectLoaded.PinMapDataModel.PowerSupplyPins
from MappingModel m in p.mappings
where BackendConstants.ExistingInstruments.Contains(m.InstrumentName)
let beforeChange = m.InstrumentName
select (m, beforeChange))
{ m.InstrumentName = string.Format("{0} (1)", beforeChange); }
#endregion
}
catch (InvalidOperationException xmlEx) { MessageBox.Show(string.Format("Error occured loading Pin Map XML.\n{0}\n{1}", xmlEx.Message, projectLoaded.PinMapFilePath)); }
catch (Exception regEx) { MessageBox.Show(regEx.Message); }
#endregion Load Pin Map

#region Relink ProjectFolder-ParentProject References
//Relink ProjectFolder ParentProject References
foreach (ProjectFolder pf in projectLoaded.ProjectTree)
{
pf.ParentProject = projectLoaded;
foreach (var obj in pf.FolderItems) //Only goes one folder layer deep, no recursion.
{
if (obj is ProjectFolder)
```

```
{
(obj as ProjectFolder).ParentProject = projectLoaded;
}
else if (obj is PaneViewModel)
{
(obj as PaneViewModel).ParentProject = projectLoaded;
(obj as PaneViewModel).ParentFolder = pf;
(obj as PaneViewModel).IsHidden = true;
}
}
}
#endregion

#region Load Project Files & Rebuild PVMs
//Load files / Rebuild PVMS
foreach (ProjectFolder pf in projectLoaded.ProjectTree)
{
switch (pf.Section)
{
case ProjectFileSection.Product_Configurations:
//pf.FolderItems.Add(new ProductConfigViewModel(typeof(ProductConfiguration), projectLoaded)
{ Header = "Production Config", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder
= pf, Icon = "Settingsd.ico" });
//pf.FolderItems.Add(new ProductConfigViewModel(typeof(ProductConfiguration), projectLoaded)
{ Header = "Development Config", IsDocument = true, IsStatic = true, IsHidden = true,
ParentFolder = pf, Icon = "Settingsd.ico" });
foreach (string hintFileRef in pf.Files)
{
string fileRef = ProjectFullPathCreator(projectLoaded, hintFileRef);
if (File.Exists(fileRef))
{
switch (Path.GetExtension(fileRef))
{
case BackendConstants.MerlinProduct_Config_Extension:
var vm = new ProductConfigViewModel(typeof(ProductConfigurationView), fileRef, projectLoaded,
pf) { IsDocument = true, IsHidden = true, Icon = "Settingsd.ico" };
pf.FolderItems.Add(vm);
break;
}
}
else { MissingProjectFilePaths.Add(fileRef); } //MessageBox.Show(string.Format("Missing file: {0}",
fileRef));
```

```
}
break;
case ProjectFileSection.Project_Data:

break;
case ProjectFileSection.Test_Limits:
if(pf.Files == null) { break; } //To prevent older projects load bug
//Load the project test limits. //Add support for delete, add new, and add existing commands...
foreach (string hintFileRef in pf.Files)
{
string fileRef = ProjectFullPathCreator(projectLoaded, hintFileRef);
if (File.Exists(fileRef))
{
switch (Path.GetExtension(fileRef))
{
case ".limits":
var vm = new TestLimitsVM(typeof(TestLimits), fileRef, projectLoaded, pf) { IsDocument = true,
IsHidden = true };
pf.FolderItems.Add(vm);
break;
case ".golds":
var goldsVM = new GoldLimitsVM(typeof(GoldLimits), fileRef, projectLoaded, pf) { IsDocument =
true, IsHidden = true };
pf.FolderItems.Add(goldsVM);
break;
case ".offsets":
var offsetsVM = new OffsetLimitsVM(typeof(OffsetLimits), fileRef, projectLoaded, pf) { IsDocument
= true, IsHidden = true };
pf.FolderItems.Add(offsetsVM);
break;
case ".traceloss":
var tracelossVM = new TracelossLimitsVM(typeof(TracelossLimits), fileRef, projectLoaded, pf) {
IsDocument = true, IsHidden = true };
pf.FolderItems.Add(tracelossVM);
break;
case ".fixtures":
var testFixturesVM = new TestFixturesVM(typeof(TestFixtures), fileRef, projectLoaded, pf) {
IsDocument = true, IsHidden = true };
pf.FolderItems.Add(testFixturesVM);
break;
}
}
```

```
else { MissingProjectFilePaths.Add(fileRef); } //MessageBox.Show(string.Format("Missing file: {0}", fileRef));
}
break;
case ProjectFileSection.Test_Parameters:
//Load the project test parameters using TestDataFileReferences as file extensions for differentiation. //Add support for delete, add new, and add existing commands...
pf.FolderItems.Add(new ConditionsViewModel(typeof(ConditionsData), projectLoaded.TestParametersFilePath, projectLoaded, pf) { Header = "Test Parameters", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder = pf });
break;
case ProjectFileSection.Test_Sequences:
//pf.FolderItems.Add(new PaneViewModel(typeof(Session_Load), projectLoaded) { Header = "Session Load", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder = pf });
//pf.FolderItems.Add(new PaneViewModel(typeof(DUT_Start_Test), projectLoaded) { Header = "DUT Start Test", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder = pf });
pf.FolderItems.Add(new DUT_TestViewModel(typeof(DUT_Test), projectLoaded.TestSequencesFilePath, projectLoaded, pf) { Header = "DUT Test", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder = pf, Icon = "TaskList.png" });
//pf.FolderItems.Add(new PaneViewModel(typeof(DUT_End_Test), projectLoaded) { Header = "DUT End Test", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder = pf });
//pf.FolderItems.Add(new PaneViewModel(typeof(SessionUnload), projectLoaded) { Header = "Session Unload", IsDocument = true, IsStatic = true, IsHidden = true, ParentFolder = pf });
break;
case ProjectFileSection.Calibration_Definitions:
foreach (string hintFileRef in (pf.Files.Count() > 0 ? pf.Files : projectLoaded.CalibrationDefinitionFileReferences))
{
string fileRef = ProjectFullPathCreator(projectLoaded, hintFileRef);
if (File.Exists(fileRef))
{
//CalDataModel cdmLoaded = CalDataModel.LoadFromJson(fileRef);
//cdmLoaded.FilePath = fileRef;
//cdmLoaded.ParentProject = projectLoaded;
//cdmLoaded.RelinkCalPointPinData(projectLoaded.PinMapDataModel);

//projectLoaded.CalibrationDefinitions.Add(cdmLoaded);
pf.FolderItems.Add(new CalibrationConfiguratorVM(typeof(CalibrationConfiguratorView), fileRef, projectLoaded, pf) { IsDocument = true, IsHidden = true });
}
else { MissingProjectFilePaths.Add(fileRef); }
}
```

```
break;
case ProjectFileSection.Calibration_Limits:
foreach (string hintFileRef in (pf.Files.Count() > 0 ? pf.Files :
projectLoaded.CalibrationLimitsFileReferences))
{
string fileRef = ProjectFullPathCreator(projectLoaded, hintFileRef);
if (File.Exists(fileRef))
{
switch (Path.GetExtension(fileRef))
{
case BackendConstants.Cal_Limit_Extension:
//CalLimitsModel clmLoaded = CalLimitsModel.LoadFromXml(fileRef);
//projectLoaded.CalibrationLimits.Add(clmLoaded);
pf.FolderItems.Add(new LimitUcVM(typeof(LimitUcEditor), fileRef, projectLoaded, pf) {
IsDocument = true, IsHidden = true });
break;
case ".xml":
//Old, Do Not Load.
break;
}
}
else { MissingProjectFilePaths.Add(fileRef); }
}
break;
case ProjectFileSection.Connection_Manager: //PinMapModel should be pre-loaded prior to this
switch case.
pf.FolderItems.Add(new PinMapViewModel(typeof(PinMap), projectLoaded.PinMapDataModel,
projectLoaded, pf) { Header = string.Format("{0} Pin Map", projectLoaded.ProjectName),
IsDocument = true, IsStatic = true, IsHidden = true, Icon = "Diagram.png" });
break;
case ProjectFileSection.Connection_Diagrams:
foreach (string hintFileRef in pf.Files)
{
string fileRef = ProjectFullPathCreator(projectLoaded, hintFileRef);
if (File.Exists(fileRef))
{
switch (Path.GetExtension(fileRef))
{
case ".xml":
var vm = new DiagramViewModel(typeof(RF_Cable_Map), fileRef, projectLoaded, pf) {
IsDocument = true, IsHidden = true };
pf.FolderItems.Add(vm);
```

```
break;
}
}
else { MissingProjectFilePaths.Add(fileRef); } //MessageBox.Show(string.Format("Missing file: {0}",
fileRef));
}
break;
case ProjectFileSection.Waveforms:
foreach (string hintFileRef in (pf.Files.Count() >  0 ? pf.Files :
projectLoaded.WaveformFileReferences))
{
string fileRef = ProjectFullPathCreator(projectLoaded, hintFileRef);
if (File.Exists(fileRef))
{
//WaveformModel wfmFile = new WaveformModel() { FilePath = fileRef };
PaneViewModel newPane = new
PaneViewModel(typeof(WaveformConverterControls.MainWindowViewModel),
typeof(WaveformConverterControls.WaveformConverterControl), fileRef, projectLoaded, pf) {
Header = Path.GetFileName(fileRef), IsDocument = true, IsHidden = true, Icon = "waveform.png"
};
//Set the paramerters to load the external control with data.
newPane.SaveOccurred += (newPane.ViewDataContext as
WaveformConverterControls.MainWindowViewModel).SaveOccuredSubscriber;
(newPane.ViewDataContext as
WaveformConverterControls.MainWindowViewModel).ChangeOccured +=
newPane.ChangeOccuredSubscriber;
//if (Path.GetExtension(fileRef) == ".aiq") //Load .aiq file format.
//{
//    // Can ignore the returned string since this code is used on project startup, no need to print to
status bar
//    (newPane.ViewDataContext as
WaveformConverterControls.MainWindowViewModel).OpenAiqFileCommandExecute(fileRef);
//}
//else //Must be .mtwf2 file format.
//{
//    // Can ignore the returned string since this code is used on project startup, no need to print to
status bar
//    (newPane.ViewDataContext as
WaveformConverterControls.MainWindowViewModel).loadMtwf(fileRef);
//}
////(newPane.ContentUserControl.DataContext as
WaveformConverterControls.MainWindowViewModel).ChangeOccured +=
```

```
newPane.ChangeOccuredSubscriber;

string eDir = MerlinProject.ExtractWaveformSubDirectoryFromFilePath(fileRef);
if (!string.IsNullOrEmpty(eDir)) //MTWF file has certified tech directory, find and place.
{

ProjectFolder sub_pf = null;
foreach (var sub_item in pf.FolderItems)
{
if (sub_item is ProjectFolder && (sub_item as ProjectFolder).FolderName == eDir)
{
sub_pf = (ProjectFolder)sub_item;
}
}
if (sub_pf == null)
{
sub_pf = new ProjectFolder() { FolderName = eDir, Section =
ProjectFileSection.WaveformSubFolder, ParentProject = pf.ParentProject }; //inherited
ProjectFileSection.Waveforms causes recursive folder bug.
pf.FolderItems.Add(sub_pf);
pf.IsExpanded = true;
}
newPane.ParentFolder = sub_pf;
sub_pf.FolderItems.Add(newPane);
sub_pf.IsExpanded = true;

}
else //AIQ, put in general waveforms.
{ newPane.ParentFolder = pf; pf.FolderItems.Add(newPane); }

//pf.ParentProject.Waveforms.Add(wfmFile);
}
else { MissingProjectFilePaths.Add(fileRef); }
}
break;
case ProjectFileSection.Digital_Patterns:
foreach (string hintFileRef in (pf.Files.Count() > 0 ? pf.Files :
projectLoaded.DigitalPatternFileReferences))
{
string fileRef = ProjectFullPathCreator(projectLoaded, hintFileRef);
if (File.Exists(fileRef))
{
```

```
switch (Path.GetExtension(fileRef))
{
case ".genericrffepattern":
pf.FolderItems.Add(new GenericRffePatternVM(typeof(GenericRffePattern), fileRef,
projectLoaded, pf) { IsDocument = true, IsHidden = true });
break;
case ".rffepat":
pf.FolderItems.Add(new DigitalPatternVM(typeof(DigitalPattern), fileRef, projectLoaded, pf) {
IsDocument = true, IsHidden = true });
break;
case ".digilevels":
#region Load custom XML fix (old)
//string oldLevelsCompileFile = Path.Combine(Path.GetDirectoryName(fileRef),
$"{Path.GetFileNameWithoutExtension(fileRef)} (Compile).digilevels");
//if (File.Exists(oldLevelsCompileFile)) //Check if old file exists.
//{
//    dlmLoaded = DigitalLevelsModel.Load_NI_Xml(oldLevelsCompileFile); //Use old xml loading.
//    dlmLoaded.FilePath = fileRef;
//    dlmLoaded.Save_NI_XML(fileRef); //Save is required now the old file is gone.
//    File.Delete(oldLevelsCompileFile);//Delete the file so this if statment does not happen again.
//}
//else
//{
//    dlmLoaded = DigitalLevelsModel.Load_NI_Xml(fileRef); //Load using new custom XML
deserialization.
//    dlmLoaded.FilePath = fileRef;
//}
#endregion Load custom XML fix
pf.FolderItems.Add(new DigitalLevelViewModel(typeof(DigitalLevel), fileRef, projectLoaded, pf) {
IsDocument = true, IsHidden = true });
break;
case ".digitiming":
#region Load custom XML fix
//string oldTimingCompileFile = Path.Combine(Path.GetDirectoryName(fileRef),
$"{Path.GetFileNameWithoutExtension(fileRef)} (Compile).digitiming");
//if (File.Exists(oldTimingCompileFile)) //Check if old file exists.
//{
//    dtmLoaded = DigitalTimingModel.Load_NI_Xml(oldTimingCompileFile); //Use old xml loading.
//    dtmLoaded.FilePath = fileRef;
//    dtmLoaded.Save_NI_XML(fileRef); //Save is required now the old file is gone.
//    File.Delete(oldTimingCompileFile); //Delete the file so this if statment does not happen again.
//}
```

```
//else
//{
//    dtmLoaded = DigitalTimingModel.Load_NI_Xml(fileRef);//Load using new custom XML
deserialization.
//    dtmLoaded.FilePath = fileRef;
//}
#endregion Load custom XML fix
pf.FolderItems.Add(new DigitalTimingVM(typeof(DigitalTiming), fileRef, projectLoaded, pf) {
IsDocument = true, IsHidden = true });
break;
}
}
else { MissingProjectFilePaths.Add(fileRef); }
}
break;
case ProjectFileSection.Dll:
foreach (string hintFileRef in pf.Files)
{
string fileRef = ProjectFullPathCreator(projectLoaded, hintFileRef);
if (File.Exists(fileRef))
{
switch (Path.GetExtension(fileRef))
{
case ".dll":
var vm = new DllFileInfoVM(typeof(DllFileInfo), fileRef, projectLoaded, pf) { IsDocument = true,
IsHidden = true};
pf.FolderItems.Add(vm);
break;
}
}
else { MissingProjectFilePaths.Add(fileRef); } //MessageBox.Show(string.Format("Missing file: {0}",
fileRef));
}
break;
default:
//if PaveViewModel
//if Project Folder
//foreach.
break;
}
}
#endregion Load Project Files & Rebuild PVMs
```

```
MissingFiles = MissingProjectFilePaths; //Set the out value for missing file paths.
return projectLoaded;
}
catch (InvalidOperationException xmlEx) { MessageBox.Show(string.Format("Error occured loading the project file.\n{0}\n{1}", xmlEx.Message, FileName)); MissingFiles = new List<string>(); return null; }
catch (Exception regEx) { MessageBox.Show(regEx.Message); MissingFiles = new List<string>(); return null;  }
}

//Helper Method.
public static string ExtractWaveformSubDirectoryFromFilePath(string FilePath)
{
//Seperate the waveforms by technology via directory seperation and put back together for project directory.
string[] dirs = FilePath.Split(Path.DirectorySeparatorChar);
string parentFolderDir = dirs[dirs.Length - 2]; //"NR", etc.
return BackendConstants.WaveformSections.Find(x => x == parentFolderDir);
}


//Experiment (Not Used)
public T GetSpecifiedSectionCollection<T>(ProjectFileSection sectionName)
{
object value = null;
switch (sectionName)
{
case ProjectFileSection.Product_Configurations:
break;
case ProjectFileSection.Test_Parameters:
break;
case ProjectFileSection.Waveforms:
case ProjectFileSection.WaveformSubFolder:
value = this.Waveforms;
break;
}

return (T)Convert.ChangeType(value, typeof(T));
}

#region ParameterValueService
```

```
public void WriteToTestParameters(ISequenceItem si, Parameter parameter)
{
try
{
object ProcessedValue = string.Empty; //Can be the value being read from or written to a test.
int valueIndex = this.ParameterMapTags.IndexOf(parameter.ParamMapping);
bool isGroup = si is IGroupable;

//Can't get or set anything with out a mapping.
if (parameter.ParamMapping == null) { parameter.TempParamMappingValue = "Not Mapped";
return; }

//Process the value and determine if it needs to be modified.
if (!parameter.IsEnum && parameter.ParamMapping.IsUnitable)
{
string modifiedValue;
Data.Services.UnitService.ValueUnitModifier(parameter.TempParamMappingValue.ToString(),
parameter.ParamUnit, parameter.ParamMapping.Unit, out modifiedValue);
ProcessedValue = modifiedValue;
}
else
{
ProcessedValue = parameter.TempParamMappingValue;
}

if (!isGroup) //Handle a UnGroupedTest
switch (parameter.Pointer)
{
case MappingPointer.Parameter:
if (parameter.TempParamMappingValue != null)
{
(si as UnGroupedTest).Test.Parameters[valueIndex].Value = ProcessedValue;
}
break;
case MappingPointer.Pin_Parameter:
(si as UnGroupedTest).Test.Parameters[valueIndex].Value =
parameter.TempParamMappingValue;
break;
default: break; //case MappingPointer.Manual: //Value Data Bound.
}
else if (isGroup) //Handle a TestGroup.
switch (parameter.Pointer)
```

```
{
case MappingPointer.Parameter:
if (parameter.TempParamMappingValue != null)
{
foreach (ITest t in (si as TestGroup).Tests)
t.Parameters[valueIndex].Value = ProcessedValue;
}
break;
case MappingPointer.Pin_Parameter:
if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the group to read
from.
foreach (ITest t in (si as TestGroup).Tests)
t.Parameters[valueIndex].Value = parameter.TempParamMappingValue;
break;
default: break; //case MappingPointer.Manual: //Value Data Bound.
}

}
catch (Exception ex) { MessageBox.Show(ex.Message); return; }
}

//Not in use.
public void ReadFromTestParameters(ISequenceItem si, Parameter parameter)
{
try
{
string modifiedValue;
object ProcessedValue = string.Empty; //Can be the value being read from or written to a test.
int valueIndex = this.ParameterMapTags.IndexOf(parameter.ParamMapping);

bool isGroup = si is IGroupable;

if (parameter.ParamMapping == null)
{
parameter.TempParamMappingValue = "Not Mapped";
return; //Can't get or set anything with out a mapping.
}

switch (parameter.Pointer)
{
case MappingPointer.Manual: //Value Data Bound.
break;
```

```
case MappingPointer.Parameter:
if (si is UnGroupedTest)
ProcessedValue = (si as UnGroupedTest).Test.Parameters[valueIndex].Value == null ?
string.Empty : (si as UnGroupedTest).Test.Parameters[valueIndex].Value.ToString();
else if (si is TestGroup)
if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the group to read
from.
ProcessedValue = (si as TestGroup).Tests[0].Parameters[valueIndex].Value == null ?
string.Empty : (si as TestGroup).Tests[0].Parameters[valueIndex].Value.ToString();

//Process the value and determine if it needs to be modified.
if (!parameter.IsEnum && parameter.ParamMapping.IsUnitable)
{
Data.Services.UnitService.ValueUnitModifier(ProcessedValue.ToString(),
parameter.ParamMapping.Unit, parameter.ParamUnit, out modifiedValue);
parameter.TempParamMappingValue = modifiedValue;
}
else
{
parameter.TempParamMappingValue = ProcessedValue;
}
break;
case MappingPointer.Pin_Parameter:

string pinName = string.Empty;
string result = string.Empty;

if (si is UnGroupedTest)
pinName = (si as UnGroupedTest).Test.Parameters[valueIndex].Value == null ? string.Empty : (si
as UnGroupedTest).Test.Parameters[valueIndex].Value.ToString();
else if (si is TestGroup)
if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the group to read
from.
pinName = (si as TestGroup).Tests[0].Parameters[valueIndex].Value == null ? string.Empty : (si as
TestGroup).Tests[0].Parameters[valueIndex].Value.ToString();

foreach (PinModel p in this.PinMapDataModel.RfPins)
if (p.PinName == pinName)
foreach (MappingModel map in p.mappings)
result += string.Format(" {0}:{1} ,", map.InstrumentName, map.IO);

parameter.TempParamMappingValue = pinName;
```

```
parameter.Value = result;
break;
}

}
catch (Exception ex) { MessageBox.Show(ex.Message); return; }
}


public void TunnelToData(ISequenceItem si, Parameter parameter, string ReadOrWrite)
{
try
{
string modifiedValue;
object ProcessedValue = string.Empty; //Can be the value being read from or written to a test.
int valueIndex = this.ParameterMapTags.IndexOf(parameter.ParamMapping);

bool isGroup = si is IGroupable;

if (parameter.ParamMapping == null)
{
parameter.TempParamMappingValue = "Not Mapped";
return; //Can't get or set anything with out a mapping.
}

switch (ReadOrWrite)
{
case "WRITE":

//Process the value and determine if it needs to be modified.
if (!parameter.IsEnum && parameter.ParamMapping.IsUnitable)
{
Data.Services.UnitService.ValueUnitModifier(parameter.TempParamMappingValue.ToString(),
parameter.ParamUnit, parameter.ParamMapping.Unit, out modifiedValue);
ProcessedValue = modifiedValue;
}
else
{
ProcessedValue = parameter.TempParamMappingValue;
}

switch (parameter.Pointer)
```

```
{
case MappingPointer.Manual: //Value Data Bound.
break;
case MappingPointer.Parameter:
if (parameter.TempParamMappingValue != null)
{
if (si is UnGroupedTest)
(si as UnGroupedTest).Test.Parameters[valueIndex].Value = ProcessedValue;
else if (si is TestGroup)
foreach (ITest t in (si as TestGroup).Tests)
t.Parameters[valueIndex].Value = ProcessedValue;
}
break;
case MappingPointer.Pin_Parameter:
if (si is UnGroupedTest)
(si as UnGroupedTest).Test.Parameters[valueIndex].Value =
parameter.TempParamMappingValue;
else if (si is TestGroup)
if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the group to read
from.
foreach (ITest t in (si as TestGroup).Tests)
t.Parameters[valueIndex].Value = parameter.TempParamMappingValue;
break;
}
break;
case "READ":
switch (parameter.Pointer)
{
case MappingPointer.Manual: //Value Data Bound.
break;
case MappingPointer.Parameter:
if (si is UnGroupedTest)
ProcessedValue = (si as UnGroupedTest).Test.Parameters[valueIndex].Value == null ?
string.Empty : (si as UnGroupedTest).Test.Parameters[valueIndex].Value.ToString();
else if (si is TestGroup)
if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the group to read
from.
ProcessedValue = (si as TestGroup).Tests[0].Parameters[valueIndex].Value == null ?
string.Empty : (si as TestGroup).Tests[0].Parameters[valueIndex].Value.ToString();

//Process the value and determine if it needs to be modified.
if (!parameter.IsEnum && parameter.ParamMapping.IsUnitable)
```

```
{
Data.Services.UnitService.ValueUnitModifier(ProcessedValue.ToString(),
parameter.ParamMapping.Unit, parameter.ParamUnit, out modifiedValue);
parameter.TempParamMappingValue = modifiedValue;
}
else
{
parameter.TempParamMappingValue = ProcessedValue;
}
break;
case MappingPointer.Pin_Parameter:

string pinName = string.Empty;
string result = string.Empty;

if (si is UnGroupedTest)
pinName = (si as UnGroupedTest).Test.Parameters[valueIndex].Value == null ? string.Empty : (si
as UnGroupedTest).Test.Parameters[valueIndex].Value.ToString();
else if (si is TestGroup)
if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the group to read
from.
pinName = (si as TestGroup).Tests[0].Parameters[valueIndex].Value == null ? string.Empty : (si as
TestGroup).Tests[0].Parameters[valueIndex].Value.ToString();

foreach (PinModel p in this.PinMapDataModel.RfPins)
if (p.PinName == pinName)
foreach (MappingModel map in p.mappings)
result += string.Format(" {0}:{1} ,", map.InstrumentName, map.IO);

parameter.TempParamMappingValue = pinName;
parameter.Value = result;
break;
}
break;
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); return; }
}



public void CheckGroupTestParametersMatch()
{
```

```
//Empty until code is transferred ?
}
#endregion

#region ITestService
public void DuplicateTest(Test testToClone)
{
var clonedTest = testToClone.Clone(this);
int insertIndex = this.Tests.IndexOf(testToClone);
this.Tests.Insert(insertIndex, clonedTest);

////Test ID management needed here.
//clonedTest.TestID = GetNewTestID();

//handle the test number situation or let the renumeration handle it????

if (string.IsNullOrEmpty(clonedTest.Parameters[4].Value.ToString())) //Must be an
UnGroupedTest.
{
//Find the SequenceItem of the test, duplicate it, and place at the same index.
ISequenceItem item = GetSequenceItemOfTest(testToClone);
var newUGTest = new UnGroupedTest()
{
ItemName = $"Test Number {clonedTest.TestNumber} - {clonedTest.TestName}",
Test = clonedTest,
Sequence = 0,
Command = item.Command,
State = item.State,
SequenceComponents = item.SequenceComponents
};
this.SequenceItems.Add(newUGTest);

}
else // Must be a TestGroup.
{
ISequenceItem item = GetSequenceItemOfTest(testToClone);

// Add the ClonedTest to the TestGroup if group is found.
(item as TestGroup).Tests.Add(clonedTest);
}
}
```

```
public void AddNewTest()
{
Test newTest = new Test(testID: GetNewTestID()) { Parameters = ProvideEmptyParameters() };

this.Tests.Add(newTest);

//Add new UnGroupedTest to the SequenceItem collection.
ISequenceItem newUGTest = new UnGroupedTest()
{
Sequence = this.SequenceItems.Count() + 1,
Command = "Execute",
SequenceComponents = new SequenceCollection<Function>(),
Test = newTest
};

this.SequenceItems.Add(newUGTest);
}

public void InsertNewTest(int insertIndex)
{
Test newTest = new Test(testID: GetNewTestID()) { Parameters = ProvideEmptyParameters() };

this.Tests.Insert((insertIndex == -1 ? this.Tests.Count() : insertIndex), newTest);

//Add new UnGroupedTest to the SequenceItem collection.
ISequenceItem newUGTest = new UnGroupedTest()
{
Sequence = this.SequenceItems.Count() + 1,
Command = "Execute",
SequenceComponents = new SequenceCollection<Function>(),
Test = newTest
};

this.SequenceItems.Add(newUGTest); //Adds to the sequence, Only inserts test to test
parameters.
}

//Switch to using the GetSequenceItemOfTest method.
public void RemoveTest(Test test)
{
this.Tests.Remove(test); //Removes from test collection.
```

```
//Remove Test from SeqeunceItem collection...
ISequenceItem UGTest = null;

//finds the test if it is in a test group.
foreach (ISequenceItem si in this.SequenceItems)
{
if (si is UnGroupedTest)
{
if ((si as UnGroupedTest).Test.TestID == test.TestID)
UGTest = si;
}
else if (si is TestGroup)
{
Test rTest = (si as TestGroup).Tests.Find(c => c.TestID == test.TestID);
if (rTest != null)
(si as TestGroup).Tests.Remove(rTest);
}
}
//If an ungrouped test is found remove it.
if (UGTest != null)
this.SequenceItems.Remove(UGTest);
}

public void RenumerateTestCollection()
{
int testnum = 1;

foreach (Test t in this.Tests)
{
if (t.IsTestNumBreak) { testnum = t.TestNumber; } //check if test is designated as a Test
Numeration Break.
t.TestNumber = testnum++;
}
}

#region Private methods

private ObservableCollection<TestParameter> ProvideEmptyParameters()
{
var testParameters = new ObservableCollection<TestParameter>();
int maxNumParameters = this.ParameterMapTags.Count; //this.Tests.Max((x) =>
x.Parameters.Count);
```

```
//Generate correct number of parameters.
for (int i = 0; i < maxNumParameters; i++)
testParameters.Add(new TestParameter() { Value = string.Empty });

return testParameters;
}

/// <summary>
/// Find and return the SequenceItem in which the given test is associated with.
/// </summary>
/// <param name="test"></param>
/// <returns></returns>
private ISequenceItem GetSequenceItemOfTest(ITest test)
{
ISequenceItem SeqItem = null;

foreach (ISequenceItem si in this.SequenceItems)
{
if (si is UnGroupedTest)
{
if ((si as UnGroupedTest).Test.TestID == test.TestID)
SeqItem = si;
}
else if (si is TestGroup)
{
ITest rTest = (si as TestGroup).Tests.Find(c => c.TestID == test.TestID);
if (rTest != null)
SeqItem = si;
}
}

return SeqItem;
}

#endregion
#endregion

#region ITestParametersService
public void AddColumn(string columnName, Type columnDataType, ParameterMapTag
paramMapTag)
{
```

```
try
{
ConditionsViewModel conditionsVM =
GetProjectFolder(ProjectFileSection.Test_Parameters).FolderItems.FirstOrDefault() as
ConditionsViewModel;
if (conditionsVM != null)
{
conditionsVM.AddColumn(columnName, paramMapTag.Unit);
this.ParameterMapTags.Add(paramMapTag);

}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

//OUTDATED.
public void ResetTable()
{
try
{
ConditionsViewModel conditionsVM =
GetProjectFolder(ProjectFileSection.Test_Parameters).FolderItems.FirstOrDefault() as
ConditionsViewModel;
if (conditionsVM != null)
{
//DataStorage.m_CSV_Conditions.Clear();
//DataStorage.m_CSV_Conditions.Columns.Clear();
conditionsVM.ResetTable(); //Resets the UI elements manually.

this.ParameterMapTags.Clear();
this.Tests.Clear();
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
#endregion

#region  ISequenceService
public void ReorderSequenceNumbers()
{
int seqNum = 1;
```

```
foreach (ISequenceItem group in this.SequenceItems)
{
group.Sequence = seqNum++;
}
}

//Temp helper method.
public Function FindAndReturnFunction(string functionName)
{
foreach (Data.DLL dll in this.DLLs)
{
foreach (Data.Function f in dll.DLLFunctions)
{
if (f.FunctionName == functionName)
{
return f;
}
}
}
return null;
}

/// <summary>
/// Builds the SequenceItems from Tests.
/// </summary>
public void BuildSequenceCollection()
{
int sequenceNumber = 1;
List<string> groups = new List<string>();
this.SequenceItems.Clear();

foreach (Test test in this.Tests)
{
int testnumber = test.TestNumber;
string testname = test.TestName;
string DefaultCommand = "Execute";
string functionCall = test.Parameters[3].Value.ToString();
string groupName = test.Parameters[4].Value.ToString();
var sequenceComponents = new SequenceCollection<Function>();

//Sets the SequenceComponents according to test function call
if (!string.IsNullOrEmpty(functionCall))
```

```
{
var function = FindAndReturnFunction(functionCall);

if (function != null)
sequenceComponents.Add(function);
else if (function is null)
{
Telerik.Windows.Controls.RadWindow SequencingTool = new
Telerik.Windows.Controls.RadWindow()
{
Header = $"Please match {functionCall} to appropriate function sequence",
Content = new UserControls.Tools.FunctionSequencingTool(this) { PassedSequenceCollection =
sequenceComponents },
WindowStartupLocation = System.Windows.WindowStartupLocation.CenterOwner,
Width = 650,
Height = 450
};
SequencingTool.ShowDialog();
}
test.Parameters[3].Value = ""; //Clears value.
}

if (string.IsNullOrEmpty(groupName))
{
var UG = new UnGroupedTest()
{
Sequence = sequenceNumber,
ItemName = $"Test Number {testnumber} - {testname}",
//TestName = testname,
//TestNumber = testnumber,
Command = DefaultCommand,
SequenceComponents = sequenceComponents,
Test = test
};

this.SequenceItems.Add(UG);
sequenceNumber++;
}
else if (!string.IsNullOrEmpty(groupName)) //!groups.Contains(groupName)
{
if (!groups.Contains(groupName))
{
```

```
var GT = new TestGroup()
{
Sequence = sequenceNumber,
ItemName = groupName,
GroupName = groupName,
Tests = new List<Test>() { test },
IsExpanded = false,
Command = DefaultCommand,
SequenceComponents = sequenceComponents
};

groups.Add(groupName); //Add string to list of pre-existing groups.
this.SequenceItems.Add(GT);
sequenceNumber++;
}
else //Adds the test to the appropriate group.
{
foreach (ISequenceItem si in this.SequenceItems)
if (si is IGroupable)
if ((si as IGroupable).GroupName == groupName)
(si as IGroupable).Tests.Add(test);
}
}
}
}

//Formerly known as "GroupAttributeCheck".
/// <summary>
/// Checks if all parameters in each group are the same
/// </summary>
public void AllTestGroupParametersCheck()
{
foreach (var si in this.SequenceItems.Where(si => si is TestGroup))
{
TestGroupParametersCheck(si as TestGroup);
}
}

/// <summary>
/// Checks the parameters of a single TestGroup.
/// </summary>
/// <param name="testGroup"></param>
```

```
public void TestGroupParametersCheck(TestGroup testGroup)
{
int numOfTests = testGroup.Tests.Count();
int sharedParams = numOfTests > 0 ? testGroup.Tests[0].Parameters.Count() : 0;

List<string> mismatchedGroups = new List<string>();
List<int> TestNumbersPerGroup = new List<int>(); //Use KvP

for (int i = 5; i < sharedParams; i++)
{
var comparable = testGroup.Tests[0].Parameters[i].Value.ToString();
for (int a = 0; a < numOfTests; a++)
{
if (testGroup.Tests[a].Parameters[i].Value.ToString() != comparable)
{
//Check if this group is NOT containted the list of mismatched groups.
if (!mismatchedGroups.Contains(testGroup.GroupName))
{
mismatchedGroups.Add(testGroup.GroupName);
}
}
}
}
//Show notification of mismatched groups if there are any.
if(mismatchedGroups.Count() != 0)
{
string errorMsg = "Mismatch in test parameters for TestGroup(s): \n";
foreach (string group in mismatchedGroups) { errorMsg += $" - {group}\n";}
MessageBox.Show(errorMsg);
}
}

//Test method w/ grouped tests handling... May be outdated.
/// <summary>
/// Compiles the sequence into a XML file using data provided in the test collection.
/// </summary>
/// <param name="filePath"></param>
public static void CompileTestSequence(string filePath, MerlinProject project)
{
using (XmlTextWriter xmlWriter = new XmlTextWriter(filePath, System.Text.Encoding.UTF8) {
Formatting = Formatting.Indented })
{
```

```
xmlWriter.WriteStartDocument();
xmlWriter.WriteStartElement("TestSequence");
xmlWriter.WriteAttributeString("Version", "1");

xmlWriter.WriteStartElement("SequenceInformation");
xmlWriter.WriteElementString("Product", "VC7643_63H380");
xmlWriter.WriteElementString("Revision", "A");
xmlWriter.WriteElementString("TestProgram", "VC7643_63H380.dll");
xmlWriter.WriteElementString("Comment", "A test program for the VC7643_63H380 product.");
xmlWriter.WriteEndElement();

xmlWriter.WriteComment("Sequence Load Steps exeucute only once at the start of the sequence
even if the sequence file is executed multiple times.");
xmlWriter.WriteStartElement("SequenceLoad");
xmlWriter.WriteElementString("SequenceLoadStep", "");
xmlWriter.WriteEndElement();

xmlWriter.WriteComment("Sequence UnLoad Steps exeucute only once at the end of the
sequence even if the sequence file is executed multiple times.");
xmlWriter.WriteStartElement("SequenceUnLoad");
xmlWriter.WriteElementString("SequenceUnLoadStep", "");
xmlWriter.WriteEndElement();

xmlWriter.WriteComment("Sequence Steps execute every time a device is tested.");
xmlWriter.WriteStartElement("Sequence");
foreach (ISequenceItem SeqItem in project.SequenceItems)
{
if (SeqItem.SequenceComponents.Count > 0)
{
foreach (ISequenceItemComponent sic in SeqItem.SequenceComponents)
{
if (sic.ItemType == SequenceItemType.Function) //Only handles functions and odes not retrive
values from SequenceControls.
{
var func = sic as Data.Function;
xmlWriter.WriteStartElement("SequenceStep");
xmlWriter.WriteAttributeString("Step", SeqItem.Command);

ITest test = new Test() { TestNumber = 0, TestResult = string.Empty }; //Make array for multiple
test numbers allowed on sequence step generation!
switch (SeqItem) //Array pos. 0 throw exception when string null maybe.
{
```

```
case UnGroupedTest _:
if ((SeqItem as UnGroupedTest).Test.TestResult.Split(':')[0] == func.FunctionName)
{
test = (SeqItem as UnGroupedTest).Test;
}
break;
case TestGroup _:
TestGroup testGroup = (SeqItem as TestGroup);
foreach (var t in testGroup.Tests.Where(t => t.TestResult.Split(':')[0] == func.FunctionName))
{
test = t;
}
break;
}

xmlWriter.WriteElementString("TestNumber", test.TestNumber.ToString());
xmlWriter.WriteElementString("TestResultMapping", test.TestResult);

xmlWriter.WriteElementString("dll", func.DLLFilePath);
xmlWriter.WriteElementString("NameSpaceAndClassName", func.DLLClassType);
xmlWriter.WriteElementString("MethodName", func.FunctionName);
xmlWriter.WriteElementString("Reprocess", "False");
xmlWriter.WriteElementString("ReprocedureName", "");

xmlWriter.WriteStartElement("MethodParameters");
foreach (Data.Parameter param in (sic as Data.Function).FunctionParameters)
{
xmlWriter.WriteStartElement("MethodParameter");
xmlWriter.WriteElementString("Parametername", param.ParamName);
xmlWriter.WriteElementString("Parametertype", param.ParamType.ToString());

//Unsure if this handles TestGroup objects well. Check.
project.TunnelToData(SeqItem, param, "READ");

xmlWriter.WriteElementString("Parametervalue", param.TempParamMappingValue.ToString());
xmlWriter.WriteEndElement();
}
xmlWriter.WriteEndElement();

xmlWriter.WriteElementString("ResultMappings", "");
xmlWriter.WriteElementString("PropertName", "");
xmlWriter.WriteElementString("PropertType", "");
```

```
xmlWriter.WriteElementString("PropertValue", "");
xmlWriter.WriteEndElement();
}
}
}
else
{
xmlWriter.WriteStartElement("SequenceStep");
xmlWriter.WriteElementString("dll", "");
xmlWriter.WriteElementString("MethodName", "");
xmlWriter.WriteElementString("MethodParameters", "");
xmlWriter.WriteElementString("PropertName", "");
xmlWriter.WriteElementString("PropertType", "");
xmlWriter.WriteElementString("PropertValue", "");
xmlWriter.WriteEndElement();
}
}
xmlWriter.WriteEndElement();

xmlWriter.WriteEndElement();
xmlWriter.WriteEndDocument();
}
}
#endregion

#region IExtension Service
public static ObservableCollection<DLL> ImportAssmblyTypes(string assemblyFilePath)
{
var types = new ObservableCollection<Data.DLL>();

try
{
Assembly asm = Helpers.AssemblyLoader.LoadWithDependencies(assemblyFilePath);

ObservableCollection<Data.DLL> DLL_Types = new ObservableCollection<Data.DLL>();

//Command now begins to start the creation of the DLL based off of the .dll
foreach (Type type in asm.GetTypes())
{
if (type.IsPublic == true && type.IsEnum != true)
{
//Gets and then sets the added functions.
```

```
var FunctionLibrary = new ObservableCollection<Function>();

foreach (MethodInfo MI in type.GetMethods(BindingFlags.Instance | BindingFlags.Public |
BindingFlags.DeclaredOnly))
{
if (MI.IsSpecialName != true && MI.IsVirtual != true)
{
//Gets the parameters of the method(s) inside the .dll
var ParamList = new ObservableCollection<Data.Parameter>();

foreach (ParameterInfo PI in MI.GetParameters())
{
Data.Parameter newParam = new Data.Parameter() { ParamName = PI.Name, ParamType =
PI.ParameterType.ToString() };

if (PI.ParameterType.IsEnum == true)
{
newParam.IsEnum = true;
newParam.ParamOptions = GetEnumValuesAsStrings(PI.ParameterType.GetEnumValues());
newParam.Pointer = MappingPointer.Parameter;
}
else
{
newParam.IsEnum = false;
newParam.Pointer = MappingPointer.Parameter;
}

ParamList.Add(newParam);
}

List<string> resultOptions = new List<string>();
if (MI.ReturnType.Name != "Void" && MI.ReturnType.Namespace == "System") //If the method
return type is not System.Void
resultOptions.Add($"{MI.Name}:{MI.ReturnType.Name}");

//Adds each field name of the MethodInfo.ReturnType if it is not included in the 'System'
namespace.
if (MI.ReturnType.Namespace != "System")
foreach (FieldInfo f_info in MI.ReturnType.GetFields(BindingFlags.Instance | BindingFlags.Public |
BindingFlags.DeclaredOnly))
resultOptions.Add($"{MI.Name}:{f_info.Name}");
```

```
FunctionLibrary.Add(new Data.Function()
{
FunctionName = MI.Name,
DLLFilePath = assemblyFilePath,
DLLClassType = type.ToString(),
NameOfDLL = System.IO.Path.GetFileName(assemblyFilePath),
FunctionParameters = ParamList,
FunctionResultOptions = resultOptions
});
}
}

DLL_Types.Add(new Data.DLL()
{
NameOfDLL = type.ToString(),
DLLFilePath = assemblyFilePath,
DLLFunctions = FunctionLibrary
});
}
}
foreach (Data.DLL dll in DLL_Types)
types.Add(dll);
}
catch (Exception Ex)
{
string errorMessage = "";

if (Ex is ReflectionTypeLoadException)
foreach (Exception e in (Ex as ReflectionTypeLoadException).LoaderExceptions)
errorMessage += string.Format("\n{0}", e.Message);
else errorMessage = Ex.Message;

MessageBox.Show(errorMessage);
return types;
}

return types;
}

private static List<string> GetEnumValuesAsStrings(Array enum_obj)
{
List<string> enum_values = new List<string>();
```

```
if (enum_obj == null) { return enum_values; } // Temporary fix, remove later
foreach (var enumVal in enum_obj)
enum_values.Add($"{enumVal}");

return enum_values;
}
#endregion

#region DataManagement Methods
//updated.
public void AddGroupAttribute(ISequenceItem sequenceItem, string groupName)
{
if (sequenceItem != null && !string.IsNullOrEmpty(groupName))
{
IGroupable group = null;
foreach (ISequenceItem g in SequenceItems)
if (g is IGroupable)
if (groupName == (g as IGroupable).GroupName)
group = (IGroupable)g;

if (group != null)
{
var testItem = (sequenceItem as UnGroupedTest).Test;
var groupItem = group.Tests[0]; //Assumes there are tests in the group.

//Check to see if the conditions/parameters match.
for (int i = 5; i < ParameterMapTags.Count; i++)
if (Convert.ToString(testItem.Parameters[i].Value) !=
Convert.ToString(groupItem.Parameters[i].Value))
{
MessageBox.Show("Group addition denied: There was a mismatch between Test Parameters.");
return; //return as to not write group data to table.
}

//If all parameters match, add it to the group.
group.Tests.Add(testItem);
}
else
{
//Create a group and add the test to that group.
SequenceItems.Insert(SequenceItems.IndexOf(sequenceItem), new TestGroup()
{
```

```
ItemName = groupName,
GroupName = groupName,
IsExpanded = true,
Sequence = sequenceItem.Sequence,
SequenceComponents = sequenceItem.SequenceComponents,
State = sequenceItem.State,
Command = sequenceItem.Command,
Tests = new List<Test>()
{
(sequenceItem as UnGroupedTest).Test
}
});
}

//Write new group information to the DataTable.
(sequenceItem as UnGroupedTest).Test.Parameters[4].Value = groupName;

//Remove old SequenceItem.
SequenceItems.Remove(sequenceItem);

}
}

//updated.
/// <summary>
/// Dissolves a TestGroup, Creates UnGroupedTest(s) from the constituent test(s), and inserts them accordingly into the sequence.
/// </summary>
/// <param name="groupName"></param>
public void DissolveGroup(TestGroup group)
{
//Creates the UnGroupedTest objects from the test(s) residing in the TestGroup.
var ungroupedTests = new ObservableCollection<UnGroupedTest>();
foreach (Test t in group.Tests)
{
ungroupedTests.Add(new UnGroupedTest()
{
ItemName = $"Test Number {t.TestNumber} - {t.TestName}",
Test = t,
Sequence = 0,
Command = group.Command,
State = group.State,
```

```
SequenceComponents = group.SequenceComponents
});

//Empty the group name parameter of the test.
t.Parameters[4].Value = string.Empty;
}

//Remove the TestGroup from collection.
int index = SequenceItems.IndexOf(group);
SequenceItems.Remove(group);

//Insert them into the sequence at the correct indexes.
foreach (UnGroupedTest t in ungroupedTests)
SequenceItems.Insert(index++, t);

}

//updated.
public  int GroupMatching(UnGroupedTest unGroupedTest, string groupName)
{
int columnCount = ParameterMapTags.Count;
var matchedTests = new List<Test>();
Test test = unGroupedTest.Test;

if (test != null)
foreach (Test rTest in Tests) //Find if there are any matching tests.
{
bool isMatchedTest = true;

//Checks to see if all the parameters match for the given test comparison.
for (int i = 5; i < columnCount; i++)
if (Convert.ToString(test.Parameters[i].Value) != Convert.ToString(rTest.Parameters[i].Value))
isMatchedTest = false;

//Add it to matchedTest list if parameters match & does not already belong to a group.
if (isMatchedTest && string.IsNullOrEmpty(rTest.Parameters[4].Value.ToString()))
matchedTests.Add(rTest);
}

if (matchedTests.Count() > 0) //If any tests were found.
{
//There are matches and none of them have assigned group names, create a new group, and
```

```
insert new group at current index.
SequenceItems.Insert(SequenceItems.IndexOf(unGroupedTest), new TestGroup()
{
ItemName = groupName,
GroupName = groupName,
IsExpanded = true,
Sequence = unGroupedTest.Sequence,
SequenceComponents = unGroupedTest.SequenceComponents,
State = unGroupedTest.State,
Command = unGroupedTest.Command,
Tests = matchedTests
});

//Write new group information to the test objects & gather the associated SequenceItem(s).
var ItemsToRemove = new List<ISequenceItem>();
foreach (ITest t in matchedTests)
{
t.Parameters[4].Value = groupName;
ItemsToRemove.AddRange(from ISequenceItem si in SequenceItems
where si is UnGroupedTest
where t.TestID == (si as UnGroupedTest).Test.TestID
select si);
}
foreach (ISequenceItem item in ItemsToRemove) //Remove old associated SequenceItem(s).
SequenceItems.Remove(item);
}

return matchedTests.Count();
}

#region Singular collection save methods
public void SaveTestItemsData(string fileName)
{
try
{
XmlSerializer xmlSerializer = new XmlSerializer(typeof(ObservableCollection<Test>));
using (StreamWriter writer = new StreamWriter(fileName))
{
xmlSerializer.Serialize(writer, this.Tests);
}
}
catch (Exception ex) { MessageBox.Show($"Error saving Test Parameters data =>
```

```
{ex.Message}"); }
}
public void SaveSequenceItemsData(string fileName)
{
try
{
ObservableCollection<SequenceItemModel> sequenceItems = new
ObservableCollection<SequenceItemModel>(); //Create a buffer collection to serialize.
foreach (var seqItem in this.SequenceItems) //Add all items to the buffer collection as
SequenceObject.
{
sequenceItems.Add(seqItem.GetModel());
}

XmlSerializer xmlSerializer = new
XmlSerializer(typeof(ObservableCollection<SequenceItemModel>));
using (StreamWriter writer = new StreamWriter(fileName))
{
xmlSerializer.Serialize(writer, sequenceItems);
}
}
catch (Exception ex) { MessageBox.Show($"Error saving DUT Test sequence data =>
{ex.Message}"); }
}
public void SaveDLLsData( string fileName)
{
try
{
XmlSerializer xmlSerializer = new XmlSerializer(typeof(ObservableCollection<DLL>));
using (StreamWriter writer = new StreamWriter(fileName))
{
xmlSerializer.Serialize(writer, this.DLLs);
}
}
catch (Exception ex) { MessageBox.Show($"Error saving DLL reference data => {ex.Message}"); }
}
public void SaveParameterMapTagData(string fileName)
{
try
{
XmlSerializer xmlSerializer = new
XmlSerializer(typeof(ObservableCollection<ParameterMapTag>));
```

```
using (StreamWriter writer = new StreamWriter(fileName))
{
xmlSerializer.Serialize(writer, this.ParameterMapTags);
}
}
catch (Exception ex) { MessageBox.Show($"Error saving Parameter Map Tag data =>
{ex.Message}"); }
}

#endregion

#region Singular collection load methods
public static void LoadTestItemsData(string fileName, MerlinProject project)
{
try
{
if (File.Exists(fileName))
{
XmlSerializer ser = new XmlSerializer(typeof(ObservableCollection<Test>));
using (StreamReader sr = new StreamReader(fileName))// Deserialize XML file.
{
project.Tests = (ObservableCollection<Test>)ser.Deserialize(sr);
}
}
}
catch (Exception ex) { MessageBox.Show($"Error loading Test Parameters data =>
{ex.Message}"); }
}
public static void LoadSequenceItemsData(string fileName, MerlinProject project)
{
try
{
if (File.Exists(fileName))
{
ObservableCollection<SequenceItemModel> sequenceItems = new
ObservableCollection<SequenceItemModel>(); //Create a buffer collection to deserialize.

XmlSerializer ser = new XmlSerializer(typeof(ObservableCollection<SequenceItemModel>));
using (StreamReader sr = new StreamReader(fileName))// Deserialize XML file.
{
sequenceItems = (ObservableCollection<SequenceItemModel>)ser.Deserialize(sr);
}
```

```
foreach (var seqItem in sequenceItems) //Add all SequenceObjects to the buffer collection as
ISequenceItem.
{
project.SequenceItems.Add(seqItem.GetISequenceItem());
}
}
}
catch (Exception ex) { MessageBox.Show($"Error loding DUT Test sequence data =>
{ex.Message}"); }
}
public static void LoadDLLsData( string fileName, MerlinProject project)
{
try
{
if (File.Exists(fileName))
{
XmlSerializer ser = new XmlSerializer(typeof(ObservableCollection<DLL>));
using (StreamReader sr = new StreamReader(fileName))// Deserialize XML file.
{
project.DLLs = (ObservableCollection<DLL>)ser.Deserialize(sr);
}
}
}
catch (Exception ex) { MessageBox.Show($"Error DLL reference data => {ex.Message}"); }
}
public static void LoadParameterMapTagData( string fileName, MerlinProject project)
{
try
{
if (File.Exists(fileName))
{
XmlSerializer ser = new XmlSerializer(typeof(ObservableCollection<ParameterMapTag>));
using (StreamReader sr = new StreamReader(fileName))// Deserialize XML file.
{
project.ParameterMapTags = (ObservableCollection<ParameterMapTag>)ser.Deserialize(sr);
}
}
}
catch (Exception ex) { MessageBox.Show($"Error Parameter Map Tag data => {ex.Message}"); }
}
```

```
#endregion

//Temp Method containing all old data relinking functionality. (Needs to be updated for null collections)
public void RelinkInternalData()
{
try
{
//Temporary Region (soon will need data relinking override void in PVM)
#region Data Relinking
//Temporary fix until object deserialization is working correctly.
//This renews the link between cloned functions on load.
foreach (ISequenceItem si in this.SequenceItems)
foreach (Data.ISequenceItemComponent ai in si.SequenceComponents)
if (ai is Data.Function)
{
var func = ai as Data.Function;
foreach (Data.DLL dll in this.DLLs)
foreach (Data.Function f in dll.DLLFunctions)
if (f.FunctionName == func.FunctionName)
if (si.ItemType != Data.SequenceItemType.Block)
func.FunctionParameters = f.FunctionParameters;
}

//Temporary fix until object deserialization is working correctly.
//This renews the link between ITests and ISequenceItems on load.
foreach (Test t in this.Tests)
foreach (ISequenceItem si in this.SequenceItems)
if (si is UnGroupedTest)
{
if ((si as UnGroupedTest).Test.TestID == t.TestID)
(si as UnGroupedTest).Test = t;
}
else if (si is TestGroup)
{
var testgroup = si as TestGroup;
bool IsTestInGroup = false;
int rIndex = -1;

foreach (Test gt in testgroup.Tests)
if (gt.TestID == t.TestID)
{
```

```
rIndex = testgroup.Tests.IndexOf(gt);
IsTestInGroup = true;
}

if (IsTestInGroup == true && rIndex > -1)
{
testgroup.Tests.RemoveAt(rIndex);
testgroup.Tests.Insert(rIndex, t);
}
}


//This renews the link between ParameterMapReference and ParamMapping in Paramters on load.
foreach (ISequenceItem si in this.SequenceItems)
{
foreach (ISequenceItemComponent c in si.SequenceComponents)
{
foreach (Parameter p in (c as Data.Function).FunctionParameters)
p.ParamMapping = this.GetParamMapTag(p.ParameterMapReference);
}
}
#endregion Data Relinking
}
catch (Exception ex) { MessageBox.Show($"Error occurred upon relinking internal data for Test Parameters and Test Sequence => {ex.Message}"); }
}

public ParameterMapTag GetParamMapTag(string parameterMapReference)
{
ParameterMapTag rpmt = null;
foreach (var pmt in this.ParameterMapTags.Where(pmt => pmt.ColumnName == parameterMapReference)) { rpmt = pmt; }
return rpmt;
}

//Used when adding a new test.
public int GetNewTestID()
{
int ID = this.Tests.Count + 1;
while (CheckIdExists(ID) == true) { ID++; }
return ID;
```

```
}

private bool CheckIdExists(int ID)
{
bool exists = false;
foreach (var _ in this.Tests.Where(t => t.TestID == ID).Select(t => new { })) { exists = true; }
return exists;
}
#endregion



#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
}
```

MTSProject.cs

```
using MerlinTest.Common;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace MerlinTest.Common.Types
{
/// <summary>
/// Class to represent individual filenames within a list. Each filename has name and if it is the
default file.
/// </summary>
```

```
[Serializable]
public class FileNameData
{
/// <summary>
/// Gets / Sets the FileName.
/// </summary>
public string FileName { get; set; }

/// <summary>
/// Gets / sets if this filename is the default.
/// </summary>
public bool IsDefault { get; set; }
}

/// <summary>
/// Class to represent the calibration definition settings and to store all calibration data and to
/// provide methods to get/set this data and to provide static import methods to import/save the
data to disk.
/// </summary>
[Serializable]
public class MTSProject
{
#region General Settings

/// <summary>
/// The name of the project the calibration data is for.
/// </summary>
public string ProjectName { get; set; }

/// <summary>
/// The name of the product the calibration data is for.
/// </summary>
public string ProductName { get; set; }

/// <summary>
/// The Version of the calibration data.
/// </summary>
public uint Version { get; set; }

/// <summary>
/// The directory the project is in.
/// </summary>
```

```
public string ProjectDirectory { get; set; }

#endregion General Settings

#region Project Files

/// <summary>
/// Filename for product configuration.
/// </summary>
string ProductConfiguration { get; set; }

/// <summary>
/// Filename for test parameters.
/// </summary>
string TestParameter { get; set; }

/// <summary>
/// Filename for test sequence.
/// </summary>
string TestSequencer { get; set; }

/// <summary>
/// List of filenames for the calibration definitions.
/// </summary>
List<FileNameData> CalibrationDefinitions { get; set; } = new List<FileNameData>();

/// <summary>
/// List of filenames for the calibration limits.
/// </summary>
List<FileNameData> CalibrationLimits { get; set; } = new List<FileNameData>();

/// <summary>
/// List of filenames for the project waveforms.
/// </summary>
List<string> Waveforms { get; set; } = new List<string>();

/// <summary>
/// List of filenames for the project connection diagrams.
/// </summary>
List<string> ConnectionDiagrams { get; set; } = new List<string>();

/// <summary>
```

```
/// PIN Map filename.
/// </summary>
string PinMap { get; set; }

/// <summary>
/// LotInfo filename.
/// </summary>
string LotInfo { get; set; }

/// <summary>
/// Digital level filename.
/// </summary>
string DigitalLevels { get; set; }

/// <summary>
/// Digital level filename.
/// </summary>
string DigitalTiming { get; set; }

/// <summary>
/// Digital pattern filename.
/// </summary>
string DigitalPattern { get; set; }

#endregion Project Files

#region Import / Save Methods

// Convert an object to a byte array
public static byte[] ObjectToByteArray(List<Object> objList)
{
List<string> tmp = new List<string>();

foreach (object item in objList)
{
if (item != null)
{
tmp.Add(item.ToString());
}
}

BinaryFormatter bf = new BinaryFormatter();
```

```
using (var ms = new MemoryStream())
{
bf.Serialize(ms, tmp);
return ms.ToArray();
}
}


/// <summary>
/// Writes the current object to XML.
/// </summary>
/// <param name="Filename">The filename and path of the file to write.</param>
public void SaveToXml(string Filename)
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(Filename))
{
xmlSerializer.Serialize(writer, this);
}
}

/// <summary>
/// Imports a MTS Project  file in the XML format.
/// </summary>
/// <param name="FileName">The filename and path of the XML based calibration file.</param>
/// <returns>A TSProject object.</returns>
public static MTSProject ImportCalDataXML(string FileName)
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(MTSProject));

MTSProject projectDataAfterDeSer;

// Deserialize XML file.
using (StreamReader sr = new StreamReader(FileName))
{
projectDataAfterDeSer = (MTSProject)ser.Deserialize(sr);
}
```

```
return projectDataAfterDeSer;
}

#endregion Import / Save Methods
}
}
```

Parameter.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.InteropServices;
using System.Xml.Serialization;
using Telerik.Windows.Documents.Spreadsheet.Expressions.Functions;
using System.Linq;
using MerlinTestStudio_Demo_Telerik.Data.Models;

namespace MerlinTestStudio_Demo_Telerik.Data
{
public interface IParameter
{
string ParamType { get; set; }
string ParamName { get; set; }
ParameterMapTag ParamMapping { get; set; }
bool IsEnum { get; set; }
MappingPointer Pointer { get; set; }
List<string> ParamOptions { get; set; }
object TempParamMappingValue { get; set; }
bool IsMapped { get; }

Parameter GetClone();

}

[TypeConverter(typeof(Helpers.EnumDescriptionTypeConverter))]
public enum MappingPointer
{
[Description("Manual")]
Manual,

[Description("Parameter")]
Parameter,
```

```
[Description("Pin Map")]
Pin_Parameter
}


/// <summary>
/// Test Parameter Object.
/// </summary>
[Serializable]
public class TestParameter : INotifyPropertyChanged
{
private object _value;

public object Value //Cannot serialize a null value with type object.
{
get { return _value; }
set { _value = value; OnPropertyChanged("Value"); }
}

public TestParameter Clone()
{
var clone = new TestParameter() { Value = this.Value };

return clone;
}

#region INPC Members

[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}


/// <summary>
/// Function Parameter Object.
```

```
/// </summary>
[Serializable]
public class Parameter : IParameter, INotifyPropertyChanged
{
private string _paramType;
private string _paramName;
private string _paramUnit;
private bool _isEnum;
private MappingPointer _pointer;
private List<string> _paramOptions = new List<string>();
private object _tempParamMappingValue;
private string _parameterMapReference;
private object _value;

/// <summary>
/// The Parameter to be placed within a function.
/// </summary>
public Parameter()
{

}

/// <summary>
/// The Type that this Parameter object contains.
/// </summary>
public string ParamType
{
get { return _paramType; }
set { _paramType = value; OnPropertyChanged("ParamType"); }
}

/// <summary>
/// The name that this Parameter object represents.
/// </summary>
public string ParamName
{
get { return _paramName; }
set { _paramName = value; OnPropertyChanged("ParamName"); }
}

public string ParameterMapReference
{
```

```
get { return _parameterMapReference; }
set
{
_parameterMapReference = value;
OnPropertyChanged("ParameterMapReference");
}
}

//public ParameterMapTag ParamMapping { get => GetParamMapTag(ParameterMapReference);
}
private ParameterMapTag _paramMapping;
[XmlIgnore]
public ParameterMapTag ParamMapping
{
get { return _paramMapping; }
set
{
_paramMapping = value;
if (value != null)
{
this.ParamUnit = value.Unit;
this.ParameterMapReference = value.ColumnName;
}

OnPropertyChanged("ParamMapping");
OnPropertyChanged("IsMapped");
}
}

public string ParamUnit
{
get { return _paramUnit; }
set { _paramUnit = value; OnPropertyChanged("ParamUnit"); }
}

public bool IsEnum
{
get { return _isEnum; }
set { _isEnum = value; OnPropertyChanged("IsEnum"); }
}
public List<string> ParamOptions
{
```

```
get { return _paramOptions; }
set { _paramOptions = value; OnPropertyChanged("ParamOptions"); }
}

public object TempParamMappingValue
{
get { return _tempParamMappingValue; }
set { _tempParamMappingValue = value; OnPropertyChanged("TempParamMappingValue"); }
}

public object Value
{
get { return _value; }
set { _value = value; OnPropertyChanged("Value"); }
}

public bool IsMapped { get => ParamMapping != null ? true : false;  }

public MappingPointer Pointer
{
get { return _pointer; }
set { _pointer = value; OnPropertyChanged("Pointer"); }
}

public Parameter GetClone()
{
Parameter clone = new Parameter()
{
ParamType = this.ParamType,
ParamName = this.ParamName,
ParameterMapReference = this.ParameterMapReference,
ParamMapping = this.ParamMapping,
ParamUnit = this.ParamUnit,
IsEnum = this.IsEnum,
ParamOptions = this.ParamOptions,
//TempParamMappingValue = this.TempParamMappingValue,
Pointer = this.Pointer
};

return clone;
;
}
```

```
public Parameter GetBlockClone()
{
Parameter clone = new Parameter()
{
ParamType = this.ParamType,
ParamName = this.ParamName,
//ParameterMapReference = this.ParameterMapReference,
ParamUnit = this.ParamUnit,
IsEnum = this.IsEnum,
ParamOptions = this.ParamOptions,
//TempParamMappingValue = this.TempParamMappingValue,
Pointer = MappingPointer.Manual
};

return clone;
;
}


#region INPC Members

[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
}
```

SequenceItem.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.IO;
```

```
using System.Linq;
using System.Runtime.Serialization;
using System.Windows;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik
{

public interface ISequenceItem
{
int Sequence { get; set; }
string ItemName { get; set; }
string ReferenceName { get; } //Shouldn't be needed with the current work flow. Remove Later.
SequenceItemType ItemType { get; }
SequenceCollection<Function>  SequenceComponents { get; set; }
string Command { get; set; }
string State { get; set; }
bool IsExpandable { get; }
bool IsExpanded { get; set; }
string Function { get;  }
List<string> TestResultOptions { get;  }


SequenceItemModel GetModel();
}

public interface IGroupable
{
string GroupName { get; set; }
bool IsGroup { get;  }
bool IsExpanded { get; set; }
List<Test> Tests { get; set; }

}

[Serializable]
public class SequenceCollection<T> : ObservableCollection<T>
{
[XmlAttribute] //Doesn't serialize.
public SequenceItemType ParentType { get; set; }
}
```

```
[Serializable]
public class SequenceBlock : ISequenceItem, INotifyPropertyChanged
{
#region Private Members
private int _sequence;
private string _itemName;
private string _command;
private string _state = "Normal";
private SequenceCollection<Function> _sequenceComponents = new
SequenceCollection<Function>() { ParentType = SequenceItemType.Block };
#endregion

#region Constructor
public SequenceBlock()
{
SequenceComponents.CollectionChanged += (sender, e) => { OnPropertyChanged("Function"); };
}
#endregion

#region Public Members
public int Sequence
{
get { return _sequence; }
set { _sequence = value; OnPropertyChanged("Sequence"); }
}
public string ItemName
{
get { return _itemName; }
set { _itemName = value; OnPropertyChanged("ItemName"); }
}
public SequenceCollection<Function> SequenceComponents
{
get { return _sequenceComponents; }
set
{
_sequenceComponents = value;
OnPropertyChanged("SequenceComponents");
}
}
public string Command
{
get { return _command; }
```

```
set { _command = value; OnPropertyChanged("Command"); }
}
public string State
{
get { return _state; }
set { _state = value; OnPropertyChanged("State"); }
}
public SequenceItemType ItemType => SequenceItemType.Block;
public bool IsExpandable => false;
public bool IsExpanded { get { return false; } set { } }
public string Function => ReturnFunctionProperty();
public string ReferenceName => ItemName;  //References self because it is an isolated block in the test sequence.
public List<string> TestResultOptions => new List<string>(); //Sequence block cannot have a result.

public string ReturnFunctionProperty()
{
if (SequenceComponents != null)
{
if (SequenceComponents.Count == 1)
{
if (SequenceComponents[0].ItemType == SequenceItemType.Function)
return (SequenceComponents[0] as Function).FunctionName.ToString();

//if (SequenceComponents[0].ItemType == SequenceItemType.DataControl)
//    return (SequenceComponents[0] as Data.DataControl).ControlName.ToString();
}
if (SequenceComponents.Count > 1) { return "Sequenced"; }
else { return "Empty"; }
}
else { return ""; }
}
#endregion

#region Model Methods
public SequenceItemModel GetModel()
{
SequenceItemModel model = new SequenceItemModel()
{
ItemType = this.ItemType,
Sequence = this.Sequence,
```

```
ItemName = this.ItemName,
SequenceComponents = this.SequenceComponents,
Command = this.Command,
State = this.State,
};

return model;
}

public static ISequenceItem GetSequenceItemFromModel(SequenceItemModel dataModel)
{
SequenceBlock block = new SequenceBlock()
{
Sequence = dataModel.Sequence,
ItemName = dataModel.ItemName,
SequenceComponents = dataModel.SequenceComponents,
Command = dataModel.Command,
State = dataModel.State,
};

return block;
}
#endregion Model Methods

#region INPC Members

[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}

[Serializable]
public class UnGroupedTest : ISequenceItem, INotifyPropertyChanged
{
#region Private Members
private int _sequence;
private string _itemName;
```

```
private string _command;
private string _state = "Normal";
private SequenceCollection<Function> _sequenceComponents = new
SequenceCollection<Function>() { ParentType = SequenceItemType.Test };
private Test _test = new Test();
private List<string> _resultOptions = new List<string>();
#endregion

#region Constructor
public UnGroupedTest()
{
SequenceComponents.CollectionChanged += (sender, e) => { OnPropertyChanged("Function");
OnPropertyChanged("TestResultOptions"); };
Test = new Test();
(Test as Test).PropertyChanged += (sender, e) => { OnPropertyChanged("ItemName"); }; //sets
everytime, remove
}
#endregion

#region Public Members
public int TestNumber
{
get { return Test.TestNumber; }
}
public string TestName
{
get { return Test.TestName; }
}
public int Sequence
{
get { return _sequence; }
set { _sequence = value; OnPropertyChanged("Sequence"); }
}
public string ItemName
{
get { return this.TestName; } //get { return $"Test Number {TestNumber} - {TestName}"; }
set { _itemName = value; OnPropertyChanged("ItemName"); } //Required by ISequenceItem
Interface.
}
public SequenceCollection<Function> SequenceComponents
{
get { return _sequenceComponents; }
```

```
set
{
_sequenceComponents = value;
SequenceComponents.CollectionChanged += (sender, e) =>
{
this.OnPropertyChanged("TestResultOptions");
this.OnPropertyChanged("Function");
};
OnPropertyChanged("SequenceComponents");
}
}
public Test Test
{
get { return _test; }
set
{
_test = value;
OnPropertyChanged("Test");
if(value != null)
{
(value as Test).PropertyChanged += (sender, e) => { OnPropertyChanged("ItemName"); };
OnPropertyChanged("ItemName");
//OnPropertyChanged("TestNumber");
//OnPropertyChanged("TestName");
}
}
}
public string Command
{
get { return _command; }
set { _command = value; OnPropertyChanged("Command"); }
}
public string State
{
get { return _state; }
set { _state = value; OnPropertyChanged("State"); }
}

public SequenceItemType ItemType => SequenceItemType.Test;
public bool IsExpandable => false;
public bool IsExpanded { get { return false; } set { } }
public string Function
```

```
{
get
{
if (SequenceComponents != null)
{
if (SequenceComponents.Count == 1)
{
if (SequenceComponents[0].ItemType == SequenceItemType.Function)
return (SequenceComponents[0] as Function).FunctionName.ToString();

//if (SequenceComponents[0].ItemType == SequenceItemType.DataControl)
//    return (SequenceComponents[0] as Data.DataControl).ControlName.ToString();
}
if (SequenceComponents.Count > 1) { return "Sequenced"; }
else { return "Empty"; }
}
else { return ""; }
}
}
public string ReferenceName => TestNumber.ToString();
public List<string> TestResultOptions
{
get
{
_resultOptions.Clear();
foreach (ISequenceItemComponent sic in SequenceComponents)
{
if(sic is IFunction)
{
foreach(string option in (sic as IFunction).FunctionResultOptions)
{
_resultOptions.Add(option);
}
}
}
return _resultOptions;
}
}
#endregion

#region Model Methods
public SequenceItemModel GetModel()
```

```
{
SequenceItemModel model = new SequenceItemModel()
{
ItemType = this.ItemType,
TestNumber = this.TestNumber,
TestName = this.TestName,
Sequence = this.Sequence,
ItemName = this.ItemName,
SequenceComponents = this.SequenceComponents,
Test = this.Test,
Command = this.Command,
State = this.State,
};

return model;
}

public static ISequenceItem GetSequenceItemFromModel(SequenceItemModel dataModel)
{
UnGroupedTest test = new UnGroupedTest()
{
//TestNumber = dataModel.TestNumber,
//TestName = dataModel.TestName,
Sequence = dataModel.Sequence,
ItemName = dataModel.ItemName,
SequenceComponents = dataModel.SequenceComponents,
Test = dataModel.Test,
Command = dataModel.Command,
State = dataModel.State,
};

return test;
}
#endregion Model Methods

#region INPC Members

[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
```

```
}

#endregion
}

[Serializable]
public class TestGroup : IGroupable, ISequenceItem, INotifyPropertyChanged
{
#region Private Members
private bool _isExpanded;
private string _groupName;
private List<Test> _tests;
private int _sequence;
private string _itemName;
private string _command;
private string _state = "Normal";
private SequenceCollection<Function> _sequenceComponents = new
SequenceCollection<Function>() { ParentType = SequenceItemType.TestGroup };
private List<string> _resultOptions = new List<string>();
#endregion

#region Constructor
public TestGroup()
{
SequenceComponents.CollectionChanged += (sender, e) => { OnPropertyChanged("Function");
OnPropertyChanged("TestResultOptions"); };
}
#endregion

#region Public Members
public int Sequence
{
get { return _sequence; }
set { _sequence = value; OnPropertyChanged("Sequence"); }
}
public string ItemName
{
get { return _itemName; }
set { _itemName = value; OnPropertyChanged("ItemName"); }
}
public SequenceCollection<Function> SequenceComponents
{
```

```
get { return _sequenceComponents; }
set
{
_sequenceComponents = value;
SequenceComponents.CollectionChanged += (sender, e) =>
{
OnPropertyChanged("Function");
OnPropertyChanged("TestResultOptions");
};
OnPropertyChanged("SequenceComponents");
}
}
public string Command
{
get { return _command; }
set { _command = value; OnPropertyChanged("Command"); }
}
public string State
{
get { return _state; }
set { _state = value; OnPropertyChanged("State"); }
}
public string GroupName
{
get { return _groupName; }
set { _groupName = value; OnPropertyChanged("GroupName"); }
}
public bool IsGroup => true;
public bool IsExpanded
{
get { return _isExpanded; }
set { _isExpanded = value; OnPropertyChanged("IsExpanded"); }
}


//Returns the test numbers in this group for visualization "1 - 19" etc.
public string TestNumber
{
get
{
if (_tests.Count == 0)
return string.Empty;
```

```
int minNumber = _tests.Min(t => t.TestNumber);
int maxNumber = _tests.Max(t => t.TestNumber);

return $"{minNumber} - {maxNumber}";
}
}

public List<Test> Tests
{
get { return _tests; }
set { _tests = value; OnPropertyChanged("Tests"); }
}
public SequenceItemType ItemType => SequenceItemType.TestGroup;
[XmlIgnore]
public bool IsExpandable => IsGroup;
[XmlIgnore]
public string Function
{
get
{
if (SequenceComponents != null)
{
if (SequenceComponents.Count == 1)
{
if (SequenceComponents[0].ItemType == SequenceItemType.Function)
return (SequenceComponents[0] as Function).FunctionName.ToString();

//if (SequenceComponents[0].ItemType == SequenceItemType.DataControl)
//    return (SequenceComponents[0] as Data.DataControl).ControlName.ToString();
}
if (SequenceComponents.Count > 1) { return "Sequenced"; }
else { return "Empty"; }
}
else { return ""; }
}
}
public string ReferenceName => GroupName;
public List<string> TestResultOptions
{
get
{
```

```
_resultOptions.Clear();
foreach (ISequenceItemComponent sic in SequenceComponents)
{
if (sic is IFunction)
{
foreach (string option in (sic as IFunction).FunctionResultOptions)
{
_resultOptions.Add(option);
}
}
}
return _resultOptions;
}
}
#endregion

#region Model Methods
public SequenceItemModel GetModel()
{
SequenceItemModel model = new SequenceItemModel()
{
ItemType = this.ItemType,
Sequence = this.Sequence,
ItemName = this.ItemName,
SequenceComponents = this.SequenceComponents,
Command = this.Command,
State = this.State,
GroupName = this.GroupName,
Tests = this.Tests
};

return model;
}

public static ISequenceItem GetSequenceItemFromModel(SequenceItemModel dataModel)
{
TestGroup testGroup = new TestGroup()
{
Sequence = dataModel.Sequence,
ItemName = dataModel.ItemName,
SequenceComponents = dataModel.SequenceComponents,
Command = dataModel.Command,
```

```
State = dataModel.State,
GroupName = dataModel.GroupName,
Tests = dataModel.Tests
};

return testGroup;
}
#endregion Model Methods

#region INPC Members

[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}

#region MerlinTestStudio DataModels initial progress
[Serializable]
public class SequenceItemModel
{

//[OptionalField(VersionAdded = 2)]
public int TestNumber { get; set; }
public string TestName { get; set; }
public int Sequence { get; set; }
public string ItemName { get; set; }
public SequenceItemType ItemType { get; set; } //No set for the actual ISequenceItems
public SequenceCollection<Function> SequenceComponents { get; set; }
public Test Test { get; set; }
public string Command { get; set; }
public string State { get; set; }
//public bool IsExpandable { get; }
//public bool IsExpanded { get; set; }
//public string Function { get; }
//public string ReferenceName { get; } //Shouldn't be needed with the current work flow. Remove
Later.
//public List<string> TestResultOptions { get; }
```

```
public string GroupName { get; set; }
//bool IsGroup { get; }
//bool IsExpanded { get; set; }
public List<Test> Tests { get; set; }

public ISequenceItem GetISequenceItem()
{
ISequenceItem seqItem = null;

switch (this.ItemType)
{
case SequenceItemType.Test:
seqItem = UnGroupedTest.GetSequenceItemFromModel(this);
break;
case SequenceItemType.TestGroup:
seqItem = TestGroup.GetSequenceItemFromModel(this);
break;
case SequenceItemType.Block:
seqItem = SequenceBlock.GetSequenceItemFromModel(this);
break;
}
seqItem.SequenceComponents.ParentType = this.ItemType;

return seqItem;
}
}

/// <summary>
/// The Model used in the GUI to represent the serializable SequenceData class
/// </summary>
public class SequenceDataModel : INotifyPropertyChanged
{
#region Private Members

private ObservableCollection<ISequenceItem> _DutSequenceItems = new
ObservableCollection<ISequenceItem>();

#endregion

public string FileName { get; set; }
public string FilePath { get; set; }
```

```
public string Revision { get; set; }


/// <summary>
/// The serializing collection of ISequenceItems translated for XML serialization.
/// </summary>
public ObservableCollection<SequenceItemModel> DutSequenceSerializable { get; set; }

/// <summary>
/// The working/GUI Interaction collection for obtaining data from the user.
/// </summary>
[XmlIgnore]
public ObservableCollection<ISequenceItem> DutSeqeunceItemsCollection
{
get { return _DutSequenceItems; }
set { _DutSequenceItems = value; OnPropertyChanged("DutSeqeunceItemsCollection"); }
}

#region Constructor
public SequenceDataModel() { }
public SequenceDataModel(string filePath,  MerlinProject parentProject)
{

}
#endregion

#region Public methods

public void SaveToXml(string filePath)
{
////Convert the ISeqenceItems to serializable models.
//foreach(ISequenceItem si in DutSeqeunceItemsCollection)
//{
//    DutSequenceSerializable.Add(si.GetSequenceItemModel());
//}
////Serialize logic here.
//using (FileStream stream = new FileStream(filePath, FileMode.OpenOrCreate))
//{
//    var xmlSerializer = new XmlSerializer(typeof(SequenceDataModel));
//    xmlSerializer.Serialize(stream, this);
//}
}
```

```
public static SequenceDataModel LoadFromXml(string filePath)
{
var loadedSDM = new SequenceDataModel();

var xmlSerializer = new XmlSerializer(typeof(SequenceDataModel));
using (FileStream stream = new FileStream(filePath, FileMode.Open))
{
loadedSDM = (SequenceDataModel)xmlSerializer.Deserialize(stream);
}

//Convert the deserialized models to the working collection of ISeqenceItems.
foreach (SequenceItemModel sim in loadedSDM.DutSequenceSerializable)
{
loadedSDM.DutSeqeunceItemsCollection.Add(sim.GetISequenceItem() as ISequenceItem);
}
return loadedSDM;
}

#endregion

#region INPC Members

[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion

}

//public class SequenceData
//{
//    public ObservableCollection<SequenceItemModel> DutSequence { get; set; }
///////
//    public string Revision { get; set; }
//    public string Product { get; set;  }
//
//
```

```
//}
#endregion MerlinTestStudio DataModels initial progress

}
```

SequenceItemComponent.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data
{

public enum SequenceItemType
{
Test,
TestGroup,
Block,

Function,
DataControl
}

public interface ISequenceItemComponent : INotifyPropertyChanged
{
string ItemName { get; set; }
SequenceItemType ItemType { get; }

/// <summary>
/// The source of this item whether it's a DataControl built into the application or a .dll reference.
/// </summary>
string ItemSource { get; set; }
string ItemReference { get; set; }

ISequenceItemComponent GetClone();
}

public interface IFunction
{
```

```
string FunctionName { get; set; }
string DLLFilePath { get; set; }
string DLLClassType { get; set; }
string NameOfDLL { get; set; }
ObservableCollection<Parameter> FunctionParameters { get; set; }
List<string> FunctionResultOptions { get; set; }

//Function Clone();
Function GetUniqueCopy();

}

[Serializable]
public class Function : ISequenceItemComponent, IFunction, ICloneable
{
private string _name;
private string _dllRef;
private string _dllClassType;
private string _nameOfDLL;
private ObservableCollection<Parameter> _functionParameters;
private List<string> _functionResultOptions;



/// <summary>
/// The name of this function.
/// </summary>
public string FunctionName
{
get { return _name; }
set { _name = value; OnPropertyChanged("FunctionName"); }
}

/// <summary>
/// The name of the DLL this function came from.
/// </summary>
public string DLLFilePath
{
get { return _dllRef; }
set { _dllRef = value; OnPropertyChanged("DLLFilePath"); }
}

public string DLLClassType
```

```
{
get { return _dllClassType; }
set { _dllClassType = value; OnPropertyChanged("DLLClassType"); }
}

public string NameOfDLL
{
get { return _nameOfDLL; }
set { _nameOfDLL = value; OnPropertyChanged("NameOfDLL"); }
}

/// <summary>
/// The collection holding the parameters of this function
/// </summary>
public ObservableCollection<Data.Parameter> FunctionParameters
{
get { return _functionParameters; }
set { _functionParameters = value; OnPropertyChanged("FunctionParameters"); }
}

public string ItemName { get; set; }
public SequenceItemType ItemType => SequenceItemType.Function;
public string ItemSource { get; set; }
public string ItemReference { get; set; }
public List<string> FunctionResultOptions
{
get { return _functionResultOptions; }
set { _functionResultOptions = value; OnPropertyChanged("FunctionResultOptions"); }
}



public object Clone()
{
Function clone = new Function()
{
FunctionName = this.FunctionName,
DLLFilePath = this.DLLFilePath,
DLLClassType = this.DLLClassType,
NameOfDLL = this.NameOfDLL,
FunctionParameters = this.FunctionParameters,
FunctionResultOptions = this.FunctionResultOptions
};
```

```
return clone;
}

public Function GetUniqueCopy()
{
Function clone = new Function()
{
FunctionName = this.FunctionName,
DLLFilePath = this.DLLFilePath,
DLLClassType = this.DLLClassType,
NameOfDLL = this.NameOfDLL,
FunctionParameters = new ObservableCollection<Parameter>(this.FunctionParameters.Select(c
=> c.GetBlockClone()).ToList()),
FunctionResultOptions = this.FunctionResultOptions
};

return clone;
}

ISequenceItemComponent ISequenceItemComponent.GetClone()
{
return (Function)Clone();
}

#region INPC Members

[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}

//Not in use.
public interface IDataControl
{
```

```
}

//Not in use.
[Serializable]
public class DataControl : ISequenceItemComponent, IDataControl
{
private string _controlName;
private System.Windows.Controls.UserControl _sequenceControl;

public DataControl()
{
ItemSource = "Merlin Test Studio";
}

public string ControlName
{
get { return _controlName; }
set { _controlName = value; OnPropertyChanged("ControlName"); }
}
public System.Windows.Controls.UserControl SequenceControl
{
get { return _sequenceControl; }
set { _sequenceControl = value; OnPropertyChanged("SequenceControl"); }
}

public string ItemName { get; set; }
public SequenceItemType ItemType { get { return SequenceItemType.DataControl; } }
public string ItemSource { get; set; }
public string ItemReference { get; set; }


ISequenceItemComponent ISequenceItemComponent.GetClone()
{
throw new NotImplementedException();
}

#region INPC Members

[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged(string propertyName)
```

```
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
}
```

TestAttribute.cs

```
using System;
using System.ComponentModel;

namespace MerlinTestStudio_Demo_Telerik.Data
{
//public enum TestAttributeType
//{
//    /// <summary>
//    /// The attribute holds information about a test.
//    /// </summary>
//    Info = 0,
//
//    /// <summary>
//    /// The attribute holds a parameter value to be used in a test.
//    /// </summary>
//    Parameter = 1,
//
//    /// <summary>
//    /// The attribute holds a limit value to be used in comparison of a test result.
//    /// (Upper & Lower is not specified currently)!
//    /// </summary>
//    Limit = 2,
//
//    /// <summary>
//    /// The attribute holds the name of a function.
//    /// </summary>
//    Functional = 3,
//
//    /// <summary>
//    /// The attribute holds the filename(string) of a waveform.
//    /// </summary>
//    Waveform = 4
//}
```

```
public interface ITestAttribute
{
string AttributeName { get; set; } //ColumnName.

TestAttributeType AttributeType { get; }

int OrderedIndex { get; set; } //ColumnIndex.

bool IsAutoGen { get; set; }

string UnitType { get; set; }

string Unit { get; set; }

bool IsUnitable { get; }
}

/// <summary>
/// Use later to handle test information (Limits, Parameters, etc.)
/// </summary>
public class TestAttribute : ITestAttribute, INotifyPropertyChanged
{
public string AttributeName { get => throw new NotImplementedException(); set => throw new
NotImplementedException(); }

public TestAttributeType AttributeType => throw new NotImplementedException();

public int OrderedIndex { get => throw new NotImplementedException(); set => throw new
NotImplementedException(); }
public bool IsAutoGen { get => throw new NotImplementedException(); set => throw new
NotImplementedException(); }
public string UnitType { get => throw new NotImplementedException(); set => throw new
NotImplementedException(); }
public string Unit { get => throw new NotImplementedException(); set => throw new
NotImplementedException(); }

public bool IsUnitable => throw new NotImplementedException();



#region INPC
[field: NonSerialized]
```

```
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
}
```

WaveformModel.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Models
{
public class WaveformModel : IFileData
{
public string FileName => Path.GetFileName(this.FilePath);
public string FilePath { get; set; }

public string ImportedFromFilePath { get; set; }

#region Events
public event Action ChangeOccured;
public event Action CommitEditBeforeSave;
#endregion

public WaveformModel()
{

}

public void Save()
{
CommitEditBeforeSave?.Invoke();
Console.WriteLine("Note: Waveforms are automatically stored in the Project File when added.");
```

```
}
public void Load()
{

}
public bool Undo()
{
Console.WriteLine("No undo operations available for Waveforms");
return false;
}

public bool Redo()
{
Console.WriteLine("No redo operations available for Waveforms");
return false;
}
}
}
```

Error.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Models.AppModels
{
//Idea to use for extending functionality to support warnings and messages as well as errors.
public interface IErrorLog
{
string Name { get; set; }
string Description { get; set; }
string Severity { get; set; } //enum
string Type { get; set; } //enum
}

public class Error
{
public Error()
{
this.ID = Guid.NewGuid();
```

```
}

public string Name { get; set; } //Or Error Code.
public string Description { get; set; }
public string Location { get; set; } //Possibly use data file path to match which PVM to set active when an error is double clicked!!!
public Guid ID { get; private set; }
}
}
```

PreferencesData.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data.Models.AppModels
{
[Serializable]
public class UserPreferencesData: INotifyPropertyChanged
{
#region Private members
private bool _isDevModeActive = false;
private string _themeName = "VisualStudio2013Theme";
#endregion

//Idea.
//public string UserSystemPreference { get; set; }

public bool IsDevModeActive
{
get { return _isDevModeActive; }
set { _isDevModeActive = value; OnPropertyChanged("IsDevModeActive"); }
}

public string Theme
{
```

```
get { return _themeName; }
set { _themeName = value; OnPropertyChanged("Theme"); }
}



public static ObservableCollection<string> ThemeOptions
{
get
{
return new ObservableCollection<string>()
{
"VisualStudio2013Theme"
};
}
}

#region Import/Save Methods
public void SaveToXml(string Filename)
{
try
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(Filename))
{
xmlSerializer.Serialize(writer, this);
}
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
}
public static UserPreferencesData LoadFromXml(string Filename)
{
try
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(UserPreferencesData));

UserPreferencesData dataAfterDeSerialize;
```

```
// Deserialize XML file.
using (StreamReader sr = new StreamReader(Filename))
{
dataAfterDeSerialize = (UserPreferencesData)ser.Deserialize(sr);
}

return dataAfterDeSerialize;
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); return null; }
}
#endregion

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
}
```

RecentData.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data.Models.AppModels
{
[Serializable]
[XmlRoot("RecentData")]
public class RecentData : INotifyPropertyChanged
{
private static int MaxRecentDataCollectionCount = 10; //Consider moving this variable to the
```

```
BackendConstants class.
private ObservableCollection<RecentProject> _recentProjects = new
ObservableCollection<RecentProject>();


public ObservableCollection<RecentProject> RecentProjects
{
get { return _recentProjects; }
set { _recentProjects = value; OnPropertyChanged("RecentProjects");
CheckAndRemoveRecentsOverflow(); }
}


public void AddRecentProject(MerlinProject project)
{
if(project == null) { return; } //If project null, do nothing.

if(RecentProjects.Count > MaxRecentDataCollectionCount)
{
CheckAndRemoveRecentsOverflow(); //Check if recents items are overflowing past the set data
collection limit.
}
List<RecentProject> rDetected = new List<RecentProject>();
foreach(RecentProject rp in RecentProjects)
{
if(rp.ProjectFilePath == project.ProjectFilePath) { rDetected.Add(rp);  }  //Mark as true if the project
being added to recents is included in recents already.
}
if (rDetected.Count > 0)
{
foreach(RecentProject rp in rDetected)
{
RecentProjects.Remove(rp);
}
}
RecentProjects.Insert(0, new RecentProject() { ProjectName = project.ProjectName,
ProjectFilePath = project.ProjectFilePath, ProjectVersion = project.Version });
}
private void CheckAndRemoveRecentsOverflow()
{
do
{
```

```
var last = RecentProjects.LastOrDefault();
if (last != null) { RecentProjects.Remove(last); }
}
while (RecentProjects.Count > MaxRecentDataCollectionCount);
}

#region Import/Save Methods
public void SaveToXml(string Filename)
{
try
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(Filename))
{
xmlSerializer.Serialize(writer, this);
}
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
}
public static RecentData LoadFromXml(string Filename)
{
try
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(RecentData));

RecentData dataAfterDeSerialize;

// Deserialize XML file.
using (StreamReader sr = new StreamReader(Filename))
{
dataAfterDeSerialize = (RecentData)ser.Deserialize(sr);
}

return dataAfterDeSerialize;
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); return null; }
}
#endregion
```

```
#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}

[Serializable]
public class RecentProject
{
public string ProjectName { get; set; }
public string ProjectFilePath { get; set; }
public double ProjectVersion { get; set; }

public string GetRecentDisplayHeader => string.Format("{0} ({1})", ProjectName, ProjectFilePath);
}
}
```

AmpConfigModel.cs

```
using MerlinTest.Common.Types;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Models
{
[Serializable]
public class AmpConfigModel : INotifyPropertyChanged
{
private RF110SrcAmpConfigurations _Config;
private bool _IsAdded;

public RF110SrcAmpConfigurations Config
```

```
{
get { return _Config; }
set { _Config = value; OnPropertyChanged("Config"); }
}
public string Name { get { return Regex.Replace(Config.ToString(), "_", " "); } }
public bool IsAdded
{
get { return _IsAdded; }
set { _IsAdded = value; OnPropertyChanged("IsAdded"); }
}

#region INPC
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
}
```

CalDataModel.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Xml.Serialization;
using System.Security.Cryptography;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System.Reflection;
using System.Collections;
using System.Windows;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using MerlinTestStudio_Demo_Telerik.Data.Models.AppModels;
```

```
using System.Collections.Specialized;
using MerlinTest.Common.Types;

namespace MerlinTestStudio_Demo_Telerik.Data.Models
{
[Serializable]
public class CalDataModel : INotifyPropertyChanged, IFileData
{
[JsonIgnore]
[XmlIgnore]
public MerlinProject ParentProject { get; set; }

/// <summary>
/// The data object this CalDataModel is based on.
/// </summary>
[JsonIgnore]
[XmlIgnore]
public CalDataV5 DataObject { get; set; }

/// <summary>
/// Gets / Sets if the limit was failed for this data-set.
/// </summary>
public bool IsLimitFailed { get; set; }

#region Private Members
private string _fileName;
private string _filePath;
private string _product;
private string _revision = "1";
private string _testProgram;
private int _calibrationDataVersion = 6;
private string _comment;
private bool _calibrateRF;
private bool _calibrateNoise;
private bool _calibrateVNA;
private double _noiseCalibrationSampleRate = 8000000; //8 MHz in Hz
private string _globalUnit = "Hz";
private string _systemVarient;
private ObservableCollection<CalPointModel> _CalPointModelCollection = new
ObservableCollection<CalPointModel>();
private ObservableCollection<AmpConfigModel> _ampConfigModelList = new
ObservableCollection<AmpConfigModel>();
```

```
#endregion

#region Public Members
public string FileName { get { return Path.GetFileNameWithoutExtension(this.FilePath); } }
public string FilePath
{
get { return _filePath; }
set { _filePath = value; OnPropertyChanged("FilePath"); OnPropertyChanged("FileName"); }
}

public string InfoToolTip
{
get { return GetInfoToolTip(); }
}

public string Product
{
get { return _product; }
set { _product = value; OnPropertyChanged("Product"); OnPropertyChanged("InfoToolTip"); }
}
public string Revision
{
get { return _revision; }
set { _revision = value; OnPropertyChanged("Revision"); OnPropertyChanged("InfoToolTip"); }
}
public string TestProgram
{
get { return _testProgram; }
set { _testProgram = value; OnPropertyChanged("TestProgram");
OnPropertyChanged("InfoToolTip"); }
}
public int CalibrationDataVersion
{
get { return _calibrationDataVersion; }
set { _calibrationDataVersion = value; OnPropertyChanged("CalibrationDataVersion"); }
}
public int NumberOfSites
{
get
{
return ParentProject.PinMapDataModel.RfPins.Count() > 0 ?
ParentProject.PinMapDataModel.RfPins.Max((x) => x.mappings.Count) : 0;
```

```
}
}
public string Comment
{
get { return _comment; }
set { _comment = value; OnPropertyChanged("Comment"); }
}

public bool CalibrateRF
{
get { return _calibrateRF; }
set { _calibrateRF = value; OnPropertyChanged("CalibrateRF"); }
}
public bool CalibrateNoise
{
get { return _calibrateNoise; }
set { _calibrateNoise = value; OnPropertyChanged("CalibrateNoise"); }
}
public bool CalibrateVNA
{
get { return _calibrateVNA; }
set { _calibrateVNA = value; OnPropertyChanged("CalibrateVNA"); }
}
public double NoiseCalibrationSampleRate
{
get { return _noiseCalibrationSampleRate; }
set { _noiseCalibrationSampleRate = value; OnPropertyChanged("NoiseCalibrationSampleRate");
}
}
public bool ResultsData { get; set; } //Indicates whether the file contains results or not.

[JsonIgnore]
[XmlIgnore]
public List<ICalResult> Results { get; set; } //For Viewer only.
[JsonIgnore]
[XmlIgnore]
public List<CalPointV5> CalPoints { get; set; } //For Viewer only.
[JsonIgnore]
[XmlIgnore]
public List<string> Rf110AmplifiersToCalibrate { get; set; } //For Viewer only.

//For Cal Config Constructor.
```

```
public string GlobalSystemVarient
{
get { return _systemVarient; }
set
{
_systemVarient = value;
OnPropertyChanged("GlobalSystemVarient"); OnPropertyChanged("InfoToolTip");
foreach (CalPointModel cpm in CalPointModelCollection) { cpm.SysPointVarient = value; }
}
}
public string GlobalUnit
{
get { return _globalUnit; }
set
{
//NoiseCalibrationSampleRate = CalPointModel.ValueUnitModifier(NoiseCalibrationSampleRate,
_globalUnit, value);
_globalUnit = value;
OnPropertyChanged("GlobalUnit");
foreach (CalPointModel cpm in CalPointModelCollection) { cpm.Unit = value; }
}
}

public ObservableCollection<CalPointModel> CalPointModelCollection
{
get { return _CalPointModelCollection; }
set
{
_CalPointModelCollection = value;
OnPropertyChanged("CalPointModelCollection");
///Incase the collection is replaced during runtime.
//if(value != null)
//{
//    HistoryManager = new DataHistoryManager(value);
//}
}
}
public ObservableCollection<AmpConfigModel> AmpConfigModelList
{
get { return _ampConfigModelList; }
set { _ampConfigModelList = value; OnPropertyChanged("AmpConfigModelList"); }
}
```

```
#endregion

#region Events

public event Action CommitEditBeforeSave;

public event Action ChangeOccured;

#endregion

#region Constructors

public CalDataModel()
{
//Empty constructor for deserialization.
DataObject = new CalDataV5();
//HistoryManager = new DataHistoryManager(this.CalPointModelCollection);
AmpConfigModelList.CollectionChanged += ItemSource_CollectionChanged;
CalPointModelCollection.CollectionChanged += CalPoint_CollectionChanged;
}

/// <summary>
/// Blank constructor for creating a new CalDataModel.
/// </summary>
public CalDataModel(MerlinProject parentProject, string userSystemTarget)
{
ParentProject = parentProject;
DataObject = new CalDataV5();
//HistoryManager = new DataHistoryManager(this.CalPointModelCollection);

GlobalUnit = "Hz";
GlobalSystemVarient = userSystemTarget;

Create_Available_AmpConfigurations();
AmpConfigModelList.CollectionChanged += ItemSource_CollectionChanged;
CalPointModelCollection.CollectionChanged += CalPoint_CollectionChanged;
}

/// <summary>
/// Import constructor for creating a CalDataModel based on an existing file.
/// </summary>
```

```
public CalDataModel(MerlinProject parentProject, CalDataV5 calDataV5, string filePath, string
userSystemTarget)
{
ParentProject = parentProject;
DataObject = calDataV5;
//HistoryManager = new DataHistoryManager(this.CalPointModelCollection);

this.FilePath = filePath;
this.Product = calDataV5.Product;
this.Revision = calDataV5.Revision;
this.TestProgram = calDataV5.TestProgram;
//this.CalibrationDataVersion = 6;
this.Comment = calDataV5.Comment;
this.CalibrateRF = calDataV5.CalibrateRF;
this.CalibrateNoise = calDataV5.CalibrateNoise;
this.CalibrateVNA = calDataV5.CalibrateVNA;
this.ResultsData = calDataV5.ResultsData;
this.NoiseCalibrationSampleRate = calDataV5.NoiseCalibrationSampleRate;

GlobalUnit = "Hz";
GlobalSystemVarient = userSystemTarget;

Create_Available_AmpConfigurations();

//Create the imported CalPoints as CalPointModels.
//foreach (CalPointV5 x in calDataV5.CalPoints) { CalPointModelCollection.Add(new
CalPointModel(x, GlobalSystemVarient, GlobalUnit)); } ///For Single Site Importing.

//Mark true the active amp configurations.
List<AmpConfigModel> ampList = AmpConfigModelList.ToList();
foreach (RF110SrcAmpConfigurations x in calDataV5.Rf110AmplifiersToCalibrate)
{
AmpConfigModel acm = ampList.Find(item => item.Config == x);
acm.IsAdded = true;
}

AmpConfigModelList.CollectionChanged += ItemSource_CollectionChanged;
CalPointModelCollection.CollectionChanged += CalPoint_CollectionChanged;
}

//Deconstructor.
~CalDataModel()
```

```
{
AmpConfigModelList.CollectionChanged -= ItemSource_CollectionChanged;
CalPointModelCollection.CollectionChanged -= CalPoint_CollectionChanged;
}
#endregion

//For Pre-made ItemSources that are not having items added or removed.
private void ItemSource_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
{
if (e.NewItems != null)
foreach (INotifyPropertyChanged item in e.NewItems)
item.PropertyChanged += this.MyType_PropertyChanged;

if (e.OldItems != null)
foreach (INotifyPropertyChanged item in e.OldItems)
item.PropertyChanged -= this.MyType_PropertyChanged;
}
private void CalPoint_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
{
if (e.NewItems != null)
{
foreach (CalPointModel item in e.NewItems)
{
item.PropertyChanged += this.MyType_PropertyChanged;
item.ValidationStatusChanged += PointStatusChanged;
}
OnChangeOccured();
}

if (e.OldItems != null)
{
foreach (CalPointModel item in e.OldItems)
{
item.PropertyChanged -= this.MyType_PropertyChanged;
item.ValidationStatusChanged -= PointStatusChanged;
}
OnChangeOccured();
}

}
private void MyType_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
```

```
OnChangeOccured();
}

#region Methods

private void Create_Available_AmpConfigurations()
{
List<AmpConfigModel> ampConfigModels = new List<AmpConfigModel>();
ampConfigModels.AddRange(from RF110SrcAmpConfigurations x in
Enum.GetValues(typeof(RF110SrcAmpConfigurations))
where x != RF110SrcAmpConfigurations.NOP
select new AmpConfigModel() { Config = x, IsAdded = false });
AmpConfigModelList = new ObservableCollection<AmpConfigModel>(ampConfigModels);
}

//public void RestoreCalPoints()
//{
//    CalPointModelCollection.Clear();
//    foreach (CalPointV5 x in DataObject.CalPoints) { CalPointModelCollection.Add(new
CalPointModel(x, GlobalSystemVarient, GlobalUnit)); }
//}

public CalDataV5 GetProcessedDataObject()
{
//Process the data for export.
DataObject.Product = this.Product;
DataObject.Revision = this.Revision;
DataObject.TestProgram = this.TestProgram;
DataObject.CalibrationDataVersion = 6;
DataObject.NumberOfSites = this.NumberOfSites;
DataObject.Comment = this.Comment;
DataObject.CalibrateRF = this.CalibrateRF;
DataObject.CalibrateNoise = this.CalibrateNoise;
DataObject.CalibrateVNA = this.CalibrateVNA;
DataObject.ResultsData = this.ResultsData;
DataObject.NoiseCalibrationSampleRate = this.NoiseCalibrationSampleRate;
//DataObject.NoiseCalibrationSampleRate =
CalPointModel.ValueUnitModifier(this.NoiseCalibrationSampleRate, GlobalUnit, "Hz");

DataObject.CalPoints.Clear();
//DataObject.CalPoints = CalPointModelCollection.Select(cpm =>
cpm.CreateCalPointV5()).ToList();
```

```
DataObject.Rf110AmplifiersToCalibrate.Clear();
DataObject.Rf110AmplifiersToCalibrate.AddRange(AmpConfigModelList.Where(x => x.IsAdded
== true).Select(x => x.Config));

return DataObject;
}
#endregion

#region Import / Save Methods
#region Binary Serialization
public void SaveToBinary(string Filename)
{
try
{
BinaryFormatter bf = new BinaryFormatter();
using (FileStream fs3 = new FileStream(Filename, FileMode.Create))
{
bf.Serialize(fs3, this);
}
}
catch(Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
}
public static CalDataModel ImportModelBinary(string MTS_CalFile)
{
try
{
CalDataModel cdm = null;
BinaryFormatter bf = new BinaryFormatter();
using (FileStream fs5 = new FileStream(MTS_CalFile, FileMode.OpenOrCreate))
{
cdm = (CalDataModel)bf.Deserialize(fs5);
}
//if(cdm != null) { cdm.HistoryManager = new DataHistoryManager(cdm.CalPointModelCollection);
} //Renew the history manager so serialized logs don't pile up in file.
return cdm;
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); return null; }
}
#endregion
#region XML Serialization
public void SaveToXml(string Filename)
```

```
{
try
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(Filename))
{
xmlSerializer.Serialize(writer, this);
}
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
}
public static CalDataModel LoadFromXml(string MTS_CalFile)
{
try
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(CalDataModel));

CalDataModel calDataModelAfterDeSerialize;

// Deserialize XML file.
using (StreamReader sr = new StreamReader(MTS_CalFile))
{
calDataModelAfterDeSerialize = (CalDataModel)ser.Deserialize(sr);
}

return calDataModelAfterDeSerialize;
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); return null; }
}
#endregion
#region  Backwards Compatability JSON Deserialization
//public void SaveToJson(string filename)
//{
//    try
//    {
//        string ser = Newtonsoft.Json.JsonConvert.SerializeObject(this, Formatting.Indented);
//        File.WriteAllText(filename, ser);
//    }
```

```
//    catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
//}
//2 Old load usages for backwards compatibility.
public static CalDataModel LoadFromJson(string MTS_CalFile)
{
try
{
CalDataModel cdm = null;
string jsonString = File.ReadAllText(MTS_CalFile);
cdm = JsonConvert.DeserializeObject(jsonString, typeof(CalDataModel)) as CalDataModel;

return cdm;
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); return null; }
}
#endregion

#endregion

private string GetInfoToolTip()
{
return string.Format("Product: {0}\nRevision: {1}\nTest Program: {2}\nTarget: {3}\nCal Data
Version: {4}", this.Product, this.Revision, this.TestProgram, this.GlobalSystemVarient,
this.CalibrationDataVersion);
}

#region Import CalDataV5 Definition

public static CalDataModel LoadFromXmlCalDataV5(string filePath, MerlinProject parentProject)
{
CalDataModel loadCdm = null;

CalDataV5 xmlCalDataObj = CalDataV5.ImportCalDataXMLNoModifyCheck(filePath);
if (xmlCalDataObj.ResultsData == true) //Results file, don't allow import.
{
MessageBox.Show("Calibration Definiton file invalid!(Calibration Results File format)");
}
else //Config file, allow import.
{
///List<string> nonExistentPins = new List<string>();
///foreach (CalPointV5 cpBase in CalDataObj.CalPoints) //Search for missing pins.
///{
```

```
///    if (PVM.ParentProject.Pins.ToList().Find(x => x.PinName == cpBase.SrcPinName) == null)
///    { nonExistentPins.Add(cpBase.SrcPinName); }
///    if (PVM.ParentProject.Pins.ToList().Find(x => x.PinName == cpBase.MeasPinName) == null)
///    { nonExistentPins.Add(cpBase.MeasPinName); }
///}
///List<string> duplicateMissingPins = nonExistentPins.GroupBy(x => x).Where(group =>
group.Count() > 1).Select(group => group.Key).ToList();
///if (duplicateMissingPins.Count() != 0)
///{
///    string missingPinsMsg = "The following pins are missing from the current PinMap: ";
///    foreach(string mps in duplicateMissingPins)
///    {
///        missingPinsMsg += string.Format("\n{0}", mps);
///    }
///    MessageBox.Show(missingPinsMsg);
///    //Need to edit the usercontrol and figure out the continuation of the import logic...
///    ///RadWindow exportView = new RadWindow()
///    ///{
///    ///    Header = "Would you like to auto-generate missing pins?",
///    ///    Height = 800,
///    ///    Width = 800,
///    ///    WindowStartupLocation = WindowStartupLocation.CenterScreen,
///    ///    Content = new UserControls.Tools.GeneratePinsFromCalibrationDefinition() {
DataContext = this }
///    ///};
///    ///exportView.Show();
///}

ObservableCollection<CalPointModel> xmlCalPointModelData = new
ObservableCollection<CalPointModel>();
if (xmlCalDataObj.CalibrationDataVersion >= 6) //Includes site numbers
{
foreach (CalPointV5 cpBase in xmlCalDataObj.CalPoints)
{
if (cpBase.IsVnaExtra == false && cpBase.SiteNumber == 1) //Requested that VnaExtras not be
imported.
{

Data.PinModel srcPin = parentProject.PinMapDataModel.RfPins.ToList().Find(x =>
x.mappings.First().IO == cpBase.SourcePort.ToString());
Data.PinModel measPin = parentProject.PinMapDataModel.RfPins.ToList().Find(x =>
x.mappings.First().IO == cpBase.MeasurePort.ToString());
```

```
CalPointModel newCpm = new CalPointModel(cpBase, parentProject.UserTargetSystem, "Hz");

int numOfSites = parentProject.PinMapDataModel.RfPins.Count() > 0 ?
parentProject.PinMapDataModel.RfPins.Max((x) => x.mappings.Count) : 0;
if (numOfSites != 0)
{
newCpm.SrcPin = srcPin;
newCpm.MeasPin = measPin;

///Check if pin-ports match for already existing name-matched pins ?

xmlCalPointModelData.Add(newCpm);
}
else //Create the CalPointModel with all other information except Src/Meas pin(s)
{
xmlCalPointModelData.Add(newCpm);
}
}
}
}
else //only includes VNA extras, no multi site generation importing
{
foreach (CalPointV5 cpBase in xmlCalDataObj.CalPoints)
{
if (cpBase.IsVnaExtra == false) //Requested that VnaExtras not be imported.
{
Data.PinModel srcPin = parentProject.PinMapDataModel.RfPins.ToList().Find(x =>
x.mappings.First().IO == cpBase.SourcePort.ToString());
Data.PinModel measPin = parentProject.PinMapDataModel.RfPins.ToList().Find(x =>
x.mappings.First().IO == cpBase.MeasurePort.ToString());

CalPointModel newCpm = new CalPointModel(cpBase, parentProject.UserTargetSystem, "Hz");

int numOfSites = parentProject.PinMapDataModel.RfPins.Count() > 0 ?
parentProject.PinMapDataModel.RfPins.Max((x) => x.mappings.Count) : 0;
if (numOfSites != 0)
{
newCpm.SrcPin = srcPin;
newCpm.MeasPin = measPin;

///Check if pin-ports match for already existing name-matched pins ?
```

```
xmlCalPointModelData.Add(newCpm);
}
else //Create the CalPointModel with all other information except Src/Meas pin(s)
{
xmlCalPointModelData.Add(newCpm);
}
}
}
}

//Set the imported data.
loadCdm = new CalDataModel(parentProject, xmlCalDataObj, filePath,
parentProject.UserTargetSystem) { Product = parentProject.ProductName, TestProgram =
parentProject.TestProgram };
foreach (var icpm in xmlCalPointModelData)
{
loadCdm.CalPointModelCollection.Add(icpm);
} //NOTE: Don't replace collection, Add imported calpoints to maintain reference to the
HistoryManager.
}

return loadCdm;
}

//CSV loading is for old cal defs
public static CalDataModel LoadFromCsvCalDataV5(string filePath, MerlinProject parentProject)
{
CalDataModel loadCdm = null;

MessageBox.Show("Warning: Pin Map (1st site only) must be built before importing, \n otherwise
non-mapped (Src & Meas port) CalPoint(s) will not be imported.");
CalDataV5 csvCalDataObj = CalDataV5.ImportCalDataCSV(filePath);
ObservableCollection<CalPointModel> CalPointModelData = new
ObservableCollection<CalPointModel>();
foreach (CalPointV5 cpBase in csvCalDataObj.CalPoints)
{
Data.PinModel srcPin = parentProject.PinMapDataModel.RfPins.ToList().Find(x =>
x.mappings.First().IO == cpBase.SourcePort.ToString());
Data.PinModel measPin = parentProject.PinMapDataModel.RfPins.ToList().Find(x =>
x.mappings.First().IO == cpBase.MeasurePort.ToString());
```

```
if (srcPin != null || measPin != null)
{
CalPointModel newCpm = new CalPointModel(cpBase, parentProject.UserTargetSystem, "Hz");

int numOfSites = parentProject.PinMapDataModel.RfPins.Count() > 0 ?
parentProject.PinMapDataModel.RfPins.Max((x) => x.mappings.Count) : 0;
if (numOfSites != 0)
{
newCpm.SrcPin = srcPin;
newCpm.MeasPin = measPin;

///Check if pin-ports match for already existing name-matched pins ?

CalPointModelData.Add(newCpm);
}
else //Create the CalPointModel with all other information except Src/Meas pin(s)
{
CalPointModelData.Add(newCpm);
}
}
}
//Set the imported data.
loadCdm = new CalDataModel(parentProject, csvCalDataObj, filePath,
parentProject.UserTargetSystem) { Product = parentProject.ProductName, TestProgram =
parentProject.TestProgram };
foreach (var icpm in CalPointModelData)
{
loadCdm.CalPointModelCollection.Add(icpm);
} //NOTE: Don't replace collection, Add imported calpoints to maintain reference to the
HistoryManager.

return loadCdm;
}

#endregion

#region Import CalDataV5 Results
/// <summary>
/// Imports calibration results and extracts bindable data types.
/// </summary>
/// <param name="calData"></param>
/// <param name="FilePath"></param>
```

```
public static CalDataModel CreateCalDataModelsFromCalDataV5(ref CalDataV5 calData, string
FilePath)
{
if (calData != null)
{
try
{
//Create the CalDataV5 Model.
CalDataModel calDataModel = new CalDataModel(null, string.Empty)
{
//FileName = Path.GetFileNameWithoutExtension(FilePath),
DataObject = calData,
FilePath = FilePath,

Product = calData.Product,
Revision = calData.Revision,
TestProgram = calData.TestProgram,
CalibrationDataVersion = calData.CalibrationDataVersion,
Comment = calData.Comment,
IsLimitFailed = calData.IsLimitFailed,

CalibrateRF = calData.CalibrateRF,
CalibrateNoise = calData.CalibrateNoise,
CalibrateVNA = calData.CalibrateVNA,
NoiseCalibrationSampleRate = calData.NoiseCalibrationSampleRate,

Results = new List<ICalResult>(),
CalPoints = calData.CalPoints,
Rf110AmplifiersToCalibrate = calData.Rf110AmplifiersToCalibrate.Select(x =>
Convert.ToString(x)).ToList()
};

//Autogenerate merged property objects for the result view binding using Reflection.
Type fieldsType = typeof(CalDataV5);
FieldInfo[] fields = typeof(CalDataV5).GetFields();
foreach (FieldInfo x in fields)
{
CalResult ResultModel = new CalResult()
{
ResultName = x.Name,
Parent = calDataModel,
CalDataModelParent = calDataModel.FileName,
```

```
CalFactorKeys = new List<ICalFactorKey>(),

KeyName = "",
ValueName = "",
KeyOrValuePropertyReferences = new List<KvP<string, string>>(),
KvPMergedObjectData = new List<object>()
};

var result = x.GetValue(calData) as IEnumerable;
foreach (var kvp in result)
{
Type valueType = kvp.GetType();

object kvpKey = valueType.GetProperty("Key").GetValue(kvp, null);
ResultModel.KeyName = kvpKey.GetType().Name;
object kvpValue = valueType.GetProperty("Value").GetValue(kvp, null);
ResultModel.ValueName = kvpValue.GetType().Name; //Attenuation results get "list'1" but it's not
needed for Cal Limit purposes.

#region Build KeyOrValuePropertyReferences
foreach (PropertyInfo KeyProp in kvpKey.GetType().GetProperties())
{
ResultModel.KeyOrValuePropertyReferences.Add(new KvP<string, string>("Key",
KeyProp.Name));
}
foreach (PropertyInfo ValueProp in kvpValue.GetType().GetProperties())
{
ResultModel.KeyOrValuePropertyReferences.Add(new KvP<string, string>("Value",
ValueProp.Name));
}
#endregion Build KeyOrValuePropertyReferences

object mergedObj = null;
if (kvpKey is double && !(kvpValue is double)) // Temp. fix for AmpGainDataResults (Key & Value
are handled during column autogeneration).
{
mergedObj = TypeMerger.Merge(kvp, kvpValue);
}
else //Handles all other result data. (Note: fine/course grain results are handled outside the loop).
{
mergedObj = TypeMerger.Merge(kvpKey, kvpValue);
}
```

```
ResultModel.KvPMergedObjectData.Add(mergedObj);
}

//If result containted any data, Add to the results list.
if (ResultModel.KvPMergedObjectData.Count() > 0)
{
calDataModel.Results.Add(ResultModel);
}
}

//Special Case: srcPortRF110CourseGrainAttenuatorResults
ICalResult courseGrain = calDataModel.Results.Find(x => x.ResultName ==
"srcPortRF110CourseGrainAttenuatorResults");
if (courseGrain != null)
{
foreach (KvP<AttenuationCalFactorKey, List<AttenuationCalFactorValue>> x in
calData.srcPortRF110CourseGrainAttenuatorResults)
{
KvPAttenuationCalFactorKeyListOfKvPOfDoubleDouble calFactor = new
KvPAttenuationCalFactorKeyListOfKvPOfDoubleDouble()
{
AmpConfiguration = Convert.ToString(x.Key.AmpConfiguration),
Frequency = x.Key.Frequency,
Group = Convert.ToString(x.Key.Group),
Coefficents = x.Key.coefficents,
Level = x.Key.Level,
Waveform = x.Key.Waveform,

//ListOfKvPOfDoubleDouble = (x.Value.Select(kv => new KvPOfDoubleDouble() { Key = kv.Key,
Value = kv.Value })).ToList()
ListOfKvPOfDoubleDouble = (x.Value.Select(kv => new KvPOfDoubleDouble() { Key =
kv.SrcAttenCalFactor, Value = kv.AttenuatorValue, PassFlag = kv.PassFlag })).ToList()
};

//Calculates the deltas on import per ListOfKvPOfDoubleDouble.
if (calFactor.ListOfKvPOfDoubleDouble.Count != 0)
{
calFactor.ListOfKvPOfDoubleDouble[0].Delta = 0; //First point has a delta of zero.
for (int i = 1; i < calFactor.ListOfKvPOfDoubleDouble.Count; i++)
{
var kv1 = calFactor.ListOfKvPOfDoubleDouble[i - 1]; //Previous point.
```

```
var kv2 = calFactor.ListOfKvPOfDoubleDouble[i]; //Delta change point.
kv2.Delta = (kv2.Key - kv1.Key); //Use key for course grain.
}
}

courseGrain.CalFactorKeys.Add(calFactor);
}
}
//Special Case: srcPortRF110FineGrainAttenuatorResults
ICalResult fineGrain = calDataModel.Results.Find(x => x.ResultName ==
"srcPortRF110FineGrainAttenuatorResults");
if (fineGrain != null)
{
foreach (KvP<AttenuationCalFactorKey, List<KvP<double, double>>> x in
calData.srcPortRF110FineGrainAttenuatorResults)
{
KvPAttenuationCalFactorKeyListOfKvPOfDoubleDouble calFactor = new
KvPAttenuationCalFactorKeyListOfKvPOfDoubleDouble()
{
AmpConfiguration = Convert.ToString(x.Key.AmpConfiguration),
Frequency = x.Key.Frequency,
Group = Convert.ToString(x.Key.Group),
Coefficents = x.Key.coefficents,
Level = x.Key.Level,
Waveform = x.Key.Waveform,

ListOfKvPOfDoubleDouble = (x.Value.Select(kv => new KvPOfDoubleDouble() { Key = kv.Key,
Value = kv.Value })).ToList()
};

//Calculates the deltas on import per ListOfKvPOfDoubleDouble.
if (calFactor.ListOfKvPOfDoubleDouble.Count != 0)
{
calFactor.ListOfKvPOfDoubleDouble[0].Delta = 0; //First point has a delta of zero.
for (int i = 1; i < calFactor.ListOfKvPOfDoubleDouble.Count; i++)
{
var kv1 = calFactor.ListOfKvPOfDoubleDouble[i - 1]; //Previous point.
var kv2 = calFactor.ListOfKvPOfDoubleDouble[i]; //Delta change point.
kv2.Delta = (kv2.Value - kv1.Value); // Use value for fine grain.
}
}
fineGrain.CalFactorKeys.Add(calFactor);
```

```
}
}

//Finally add the CalDataModel to the master results list.
//CalibrationResults.Add(calDataModel);
return calDataModel;
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
return null;
}
}
else
{
Console.WriteLine("CalDataV5 reference data parameter is null...");
return null;
}
}
#endregion

#region IUndoable Implementation
[JsonIgnore]
[XmlIgnore]
private List<HistoryRecord> historyLog = new List<HistoryRecord>();
[JsonIgnore]
[XmlIgnore]
private bool undoMode;

public bool Undo()
{
if (historyLog.Count == 0)
{
return false;
}
else
{
//Check historyLog.Count() and remove the first item(s) while count is above the set LIMIT.
}
undoMode = true;
```

```
HistoryRecord current = historyLog[historyLog.Count - 1];
historyLog.RemoveAt(historyLog.Count - 1);

PerformUndoAction(current);

undoMode = false;

return true;
}

//Contains the undo logic.
private void PerformUndoAction(HistoryRecord current)
{
switch (current.Action)
{
case HistoryAction.Delete:
if (current.DataItem is List<KeyValuePair<int, object>>)
{
List<KeyValuePair<int, object>> items = new List<KeyValuePair<int, object>>((current.DataItem
as List<KeyValuePair<int, object>>).OrderBy(x => x.Key));
for (int i = 0; i < items.Count; i++)
{
CalPointModel revised = items[i].Value as CalPointModel;
revised.SysPointVarient = GlobalSystemVarient; revised.Unit = GlobalUnit;
CalPointModelCollection.Insert(items[i].Key, revised);
this.PointStatusChanged(items[i].Value as CalPointModel); //Temp handles errors accordingly if
it's not contained in the collection.
}
}
break;
case HistoryAction.Add:
if(current.DataItem is List<KeyValuePair<int, object>>)
{
List<KeyValuePair<int, object>> items = new List<KeyValuePair<int, object>>((current.DataItem
as List<KeyValuePair<int, object>>).OrderByDescending(x => x.Key));
for (int i = 0; i < items.Count; i++)
{
CalPointModelCollection.Remove(items[i].Value as CalPointModel);
this.PointStatusChanged(items[i].Value as CalPointModel); //Temp handles errors accordingly if
contained in the collection.
}
}
```

```
break;
case HistoryAction.Duplicate:
if (current.DataItem is List<KeyValuePair<int, object>>)
{
List<KeyValuePair<int, object>> items = new List<KeyValuePair<int, object>>((current.DataItem
as List<KeyValuePair<int, object>>).OrderByDescending(x => x.Key));
for (int i = 0; i < items.Count; i++)
{
CalPointModelCollection.Remove(items[i].Value as CalPointModel);
}
}
break;
case HistoryAction.Edit:
if (current.DataItem is List<Edit<int, object, string>>)
{
List<Edit<int, object, string>> editedItems = current.DataItem as List<Edit<int, object, string>>;
foreach (Edit<int, object, string> edit in editedItems)
{
object dataItem = CalPointModelCollection[edit.Index];
PropertyInfo cellProp = dataItem.GetType().GetProperty(edit.Property);
cellProp.SetValue(dataItem, edit.Value);
}
}
break;
default:
break;
}
}

//Subroutine for pre-checks and adding to the HistoryRecord collection.
private void LogAction(HistoryRecord record)
{
if (!undoMode)
{
historyLog.Add(record);
}
}

public void LogEditAction(List<Edit<int, object, string>> editedItems)
{
LogAction(new HistoryRecord()
{
```

```
Action = HistoryAction.Edit,
DataItem = editedItems
});
}

public void LogInsertAction(List<KeyValuePair<int, object>> insertedItems)
{
LogAction(new HistoryRecord()
{
Action = HistoryAction.Add,
DataItem = insertedItems
});
}

public void LogDuplicateAction(List<KeyValuePair<int, object>> duplicatedItems)
{
LogAction(new HistoryRecord()
{
Action = HistoryAction.Duplicate,
DataItem = duplicatedItems
});
}

public void LogDeleteAction(List<KeyValuePair<int, object>> deletedItems)
{
LogAction(new HistoryRecord()
{
Action = HistoryAction.Delete,
DataItem = deletedItems
});
}
#endregion

//Event subscriber to add or remove errors from error logs.
public void PointStatusChanged(CalPointModel cpm)
{
switch (cpm.PointStatus)
{
case PointValidationStatus.Invalid:
if (this.CalPointModelCollection.Contains(cpm))
{
var errToAdd = new Error() { Name = "CalPoint Error", Description = cpm.ToolTipString, Location
```

```
= Path.GetFileName(this.FilePath) };
cpm.ErrorIDs.Add(errToAdd.ID); //Add the error ID to the cal point we have the specific error tracked.

DataStorage.ErrorLogs.Add(errToAdd); //Add it to the pane source.
}
else
{
RemoveAllErrors(cpm);
}
break;
case PointValidationStatus.Valid:
RemoveAllErrors(cpm);
break;
case PointValidationStatus.Warning:
//Create logic when there are multiple types of errors.
break;
}
}
//Temp Method to remove all CalPoint Errors
public void RemoveAllErrors(CalPointModel cpm)
{
List<Error> errorsFound = new List<Error>();
foreach (Error e in DataStorage.ErrorLogs)
{
if (cpm.ErrorIDs.Contains(e.ID))
{
errorsFound.Add(e);
}
}
if (errorsFound.Count() > 0)
{
foreach (var errFound in errorsFound)
{
DataStorage.ErrorLogs.Remove(errFound);
cpm.ErrorIDs.Remove(errFound.ID);
}
}
}

//IFileData
public void Save()
```

```
{
CommitEditBeforeSave?.Invoke();
this.SaveToXml(this.FilePath);
Console.WriteLine(string.Format("Saved: {0}", this.FilePath));
}
public void Load()
{

}
public bool Redo()
{
Console.WriteLine("Redo operation not implemented for Calibration Definitions");
return true;
}



#region INPC
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
OnChangeOccured();
}
#endregion

public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
}
}
```

CalFactorKey.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace MerlinTestStudio_Demo_Telerik.Data.Models
{

public interface ICalFactorKey { }

#region Attenuation
[Serializable]
public class KvPAttenuationCalFactorKeyListOfKvPOfDoubleDouble : ICalFactorKey
{
#region Key
public string AmpConfiguration { get; set; }
public string Group { get; set; }
public double Frequency { get; set; }
public List<double> Coefficents { get; set; }
public string Waveform { get; set; }
public double Level { get; set; }
#endregion

#region Value
public List<KvPOfDoubleDouble> ListOfKvPOfDoubleDouble { get; set; }
#endregion
}
[Serializable]
public class KvPOfDoubleDouble : INotifyPropertyChanged
{
private double _regressed;

public double Key { get; set; }
public double Value { get; set; }
public double Regressed
{
get { return _regressed; }
set { _regressed = value; OnPropertyChanged("Regressed"); }
}
public double Delta { get; set; }

public bool PassFlag { get; set; }

#region INPC
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
```

```
public void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
#endregion

}
```

CalLimitsModel.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Newtonsoft.Json;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Dynamic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Runtime.Serialization.Formatters.Binary;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Xml;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data.Models.CalModels
{
[Serializable]
public class CalLimitsModel : INotifyPropertyChanged, IFileData
{
public const double DefaultMinTolerance = 3;
public const double DefaultMaxTolerance = 3;
```

```
#region Private Members
private string _filePath;
private ObservableCollection<ICalResult> _results = new ObservableCollection<ICalResult>();
//private LimitUc _limitUcObj = new LimitUc();
#endregion

#region Public Members
public string FileName => Path.GetFileNameWithoutExtension(this.FilePath);
public string FilePath { get { return _filePath; } set { _filePath = value;
OnPropertyChanged("FilePath"); OnPropertyChanged("FileName"); } }

public double CalibrationDataVersion { get; set; }
public byte[] Hash { get; set; } //Hash for the Cal Limits
public byte[] HashCalDef { get; set; } //Hash for the Cal Data that produced the Limits.
public ObservableCollection<ICalResult> Results
{
get { return _results; }
set { _results = value; OnPropertyChanged("Results"); }
}

//The compile format for the cal limits. Model data is converted to this for serialization.
//public LimitUc LimitUcObj
//{
//    get { return _limitUcObj; }
//    set { _limitUcObj = value; OnPropertyChanged("TestLimitObj"); }
//}
#endregion

#region Actions and Events
public event Action ChangeOccured;
public event Action CommitEditBeforeSave;
#endregion

#region Constructor
public CalLimitsModel() { }
public CalLimitsModel(string filePath)
{
this.FilePath = filePath;
}
public CalLimitsModel(string filePath, CalDataModel calDataModelBase)
{
this.FilePath = filePath;
```

```
//resultBase interpolation to limits. don't store. Just reference.
}
#endregion

public void SetLimitsTolerance(CalResult result, double MinTolerance, double MaxTolerance)
{
try
{
foreach (dynamic exObj in result.KvPMergedObjectData)
{
//Use the LimitMinMax store to calculate the upper and lower limits for CalFactor and ENR
properties.
var dictionary = (IDictionary<string, object>)exObj;

if(!Double.IsNaN(double.Parse(exObj.MinTolerance.ToString())) ||
!Double.IsNaN(double.Parse(exObj.MinTolerance.ToString())))
{
exObj.MinTolerance = MinTolerance;
exObj.MaxTolerance = MaxTolerance;

exObj.Min = (double)exObj.Baseline - MinTolerance;
exObj.Max = (double)exObj.Baseline + MaxTolerance;
}
else
{
exObj.Min = double.NaN;
exObj.Max = double.NaN;
exObj.MinTolerance = double.NaN;
exObj.MaxTolerance = double.NaN;
}
//exObj.MinTolerance = MinTolerance;
//exObj.MaxTolerance = MaxTolerance;

//exObj.Min = (double)exObj.Baseline - MinTolerance;
//exObj.Max = (double)exObj.Baseline + MaxTolerance;
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); Console.WriteLine(ex.Message); }

}

#region XML Import/Export
```

```
public static CalLimitsModel LoadFromXml(string FileName)
{
try
{
UserCalibrationLimits limitsFile = UserCalibrationLimits.ImportLimitUcXML(FileName); //Load the LimitsUc File.
CalLimitsModel limitsModel = new CalLimitsModel(FileName) //Create a new model to return after processing.
{
Hash = limitsFile.Hash,
HashCalDef = limitsFile.HashCalDef, //Only set on creation. This allows data continuity.
CalibrationDataVersion = limitsFile.CalibrationDataVersion
};

//Merge the KeyValuePair objects into one object.
Type fieldsType = typeof(UserCalibrationLimits);
FieldInfo[] fields = typeof(UserCalibrationLimits).GetFields();
foreach (FieldInfo x in fields)
{
CalResult ResultModel = new CalResult()
{
ResultName = x.Name,
//Parent = calDataModel,
//CalDataModelParent = calDataModel.FileName,
CalFactorKeys = new List<ICalFactorKey>(),

KeyName = "", //NOT SERIALIZED!!!
ValueName = "", //Not needed since value name is known "MinMaxData".
KeyOrValuePropertyReferences = new List<KvP<string, string>>(),
KvPMergedObjectData = new List<object>()
};

var result = x.GetValue(limitsFile) as IEnumerable;
foreach (var kvp in result)
{
Type valueType = kvp.GetType();

object kvpKey = valueType.GetProperty("Key").GetValue(kvp, null);
ResultModel.KeyName = kvpKey.GetType().Name; //SETS TOO MUCH!!!
object kvpValue = valueType.GetProperty("Value").GetValue(kvp, null);
ResultModel.ValueName = kvpValue.GetType().Name; //SETS TOO MUCH!!! //Attenuation results get "list'1" but it's not needed for Cal Limit purposes.
```

```
#region Build KeyOrValuePropertyReferences
foreach (PropertyInfo KeyProp in kvpKey.GetType().GetProperties())
{
ResultModel.KeyOrValuePropertyReferences.Add(new KvP<string, string>("Key",
KeyProp.Name));
}
foreach (PropertyInfo ValueProp in kvpValue.GetType().GetProperties())
{
ResultModel.KeyOrValuePropertyReferences.Add(new KvP<string, string>("Value",
ValueProp.Name));
}
#endregion Build KeyOrValuePropertyReferences

object mergedObj = null;
///if (kvpKey is double && !(kvpValue is double)) // Temp. fix for AmpGainDataResults (Key &
Value are handled during column autogeneration).
///{
///    mergedObj = TypeMerger.Merge(kvp, kvpValue);
///}
///else //Handles all other result data. (Note: fine/course grain results are handled outside the
loop).
///{
///    mergedObj = TypeMerger.Merge(kvpKey, kvpValue);
///}
mergedObj = TypeMerger.Merge(kvpKey, kvpValue);

ResultModel.KvPMergedObjectData.Add(mergedObj);
}

//If result containted any data, Add to the results list.
if (ResultModel.KvPMergedObjectData.Count() > 0)
{
limitsModel.Results.Add(ResultModel);
}
}

//Convert the Objects with ExpandoObjects (so they can be edited)
foreach (CalResult calresult in limitsModel.Results)
{
//Create the ExpandoObjects using the objects from the CalResult, then replace the data.
List<dynamic> newMergedObjData = new List<dynamic>();
```

```
foreach (dynamic mergedObj in calresult.KvPMergedObjectData)
{
var exObj = new ExpandoObject();
var dictionary = (IDictionary<string, object>)exObj;

foreach (var property in mergedObj.GetType().GetProperties())
{
dictionary.Add(property.Name, property.GetValue(mergedObj));
}
newMergedObjData.Add(exObj);
}
calresult.KvPMergedObjectData = newMergedObjData; //Replace the objects with
ExpandoObjects. Key & Value References stay the same.
}

return limitsModel;
}
catch (Exception ex)
{
//System.Windows.MessageBox.Show(ex.Message);
Console.WriteLine($"Calibration Limit File: {ex.Message}");
return null;
}
}

//LATEST
public void SaveToXml(string fileName)
{
UserCalibrationLimits limitsFile = new UserCalibrationLimits();

//Build UserCalibrationLimits class from model data using reflection.
try
{
limitsFile.CalibrationDataVersion = this.CalibrationDataVersion;
limitsFile.HashCalDef = this.HashCalDef;

foreach (CalResult calresult in this.Results)
{
var field = typeof(UserCalibrationLimits).GetField(calresult.ResultName);
var listType = field.FieldType;
var keyType = listType.GetGenericArguments()[0].GetGenericArguments()[0];
var valueType = listType.GetGenericArguments()[0].GetGenericArguments()[1];
```

```
var list = (IList)Activator.CreateInstance(listType);
foreach (ExpandoObject item in calresult.KvPMergedObjectData)
{
var key = Activator.CreateInstance(keyType);
var value = Activator.CreateInstance(valueType);
foreach (var property in (IDictionary<String, Object>)item)
{
var kvpRef = calresult.KeyOrValuePropertyReferences.Find(x => x.Value == property.Key);
//DOES NOT DESERIALIZE BACK IN FOR RE-USE!!!
if (kvpRef.Key == "Key")
{
PropertyInfo keyProp = keyType.GetProperty(property.Key);
keyProp.SetValue(key, property.Value);
}
}
foreach (var property in (IDictionary<String, Object>)item)
{
PropertyInfo valueProp = null;
switch (property.Key)
{
case "Min":
case "MinTolerance":
case "Max":
case "MaxTolerance":
valueProp = valueType.GetProperty(property.Key);
valueProp.SetValue(value, property.Value);
break;
case "CalFactor":
case "ENR":
case "Baseline":
valueProp = valueType.GetProperty("Baseline"); //Only in MinMaxData class.
valueProp.SetValue(value, property.Value);
break;
}
}
var keyValuePair = Activator.CreateInstance(listType.GetGenericArguments()[0], key, value);
list.Add(keyValuePair);
}
field.SetValue(limitsFile, list);
}
```

```
limitsFile.SaveToXml(fileName);
}
catch (Exception ex)
{
// Handle exception
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}

}

///shorter code, needs testing
//public static CalLimitsModel LoadFromXml(string FileName)
//{
//    UserCalibrationLimits limitsFile = UserCalibrationLimits.ImportLimitUcXML(FileName);
//    CalLimitsModel limitsModel = new CalLimitsModel(FileName) { Hash = limitsFile.Hash,
CalibrationDataVersion = limitsFile.CalibrationDataVersion };
//
//    foreach (FieldInfo x in typeof(UserCalibrationLimits).GetFields())
//    {
//        var result = x.GetValue(limitsFile) as IEnumerable;
//        if (result == null) continue;
//
//        CalResult ResultModel = new CalResult()
//        {
//            ResultName = x.Name,
//            CalFactorKeys = new List<ICalFactorKey>(),
//            KeyOrValuePropertyReferences = new List<KvP<string, string>>(),
//            KvPMergedObjectData = new List<object>()
//        };
//
//        foreach (var kvp in result)
//        {
//            Type valueType = kvp.GetType();
//            object kvpKey = valueType.GetProperty("Key").GetValue(kvp, null);
//            object kvpValue = valueType.GetProperty("Value").GetValue(kvp, null);
//
//            foreach (PropertyInfo prop in kvpKey.GetType().GetProperties())
//            {
//                ResultModel.KeyOrValuePropertyReferences.Add(new KvP<string, string>("Key",
prop.Name));
//            }
```

```
//            foreach (PropertyInfo prop in kvpValue.GetType().GetProperties())
//            {
//                ResultModel.KeyOrValuePropertyReferences.Add(new KvP<string, string>("Value",
prop.Name));
//            }
//
//            object mergedObj = TypeMerger.Merge(kvpKey, kvpValue);
//            ResultModel.KvPMergedObjectData.Add(mergedObj);
//        }
//
//        if (ResultModel.KvPMergedObjectData.Count() > 0)
//        {
//            limitsModel.Results.Add(ResultModel);
//        }
//    }
//
//    foreach (CalResult calresult in limitsModel.Results)
//    {
//        List<dynamic> newMergedObjData = new List<dynamic>();
//        foreach (dynamic obj in calresult.KvPMergedObjectData)
//        {
//            dynamic newMergedObj = new ExpandoObject();
//            foreach (PropertyInfo prop in obj.GetType().GetProperties())
//            {
//                ((IDictionary<string, object>)newMergedObj).Add(prop.Name, prop.GetValue(obj,
null));
//            }
//            newMergedObjData.Add(newMergedObj);
//        }
//        calresult.KvPMergedObjectData = newMergedObjData;
//    }
//
//    return limitsModel;
//}


///public void Save_Custom_XML(string FileName)
///{
///    try
///    {
///        string MaxMinDataTypeName = typeof(MaxMinData).Name;
///        using (XmlTextWriter xmlWriter = new XmlTextWriter(FileName,
```

```
System.Text.Encoding.UTF8) { Formatting = System.Xml.Formatting.Indented })
///       {
///           xmlWriter.WriteStartDocument();
///           xmlWriter.WriteStartElement(typeof(UserCalibrationLimits).Name); //Change to name of
type
///           xmlWriter.WriteAttributeString("xmlns:xsi", "http://www.w3.org/2001/XMLSchema-
instance");
///           xmlWriter.WriteAttributeString("xmlns:xsd", "http://www.w3.org/2001/XMLSchema");
///           xmlWriter.WriteElementString("CalibrationDataVersion",
this.CalibrationDataVersion.ToString());
///
///           // Make sure the Hash field is NULL before we compute the object as an array of bytes.
///           this.Hash = null;
///
///           // Populate list of objects used to calculate MD5 Hash
///           List<object> LimitUcPropertiesToHash = SavePropertiesToBeHashed();
///
///           // Treat the object as a series of bytes.
///           var objectInBytes = LimitUcPropertiesToHash.SelectMany(s =>
System.Text.Encoding.UTF8.GetBytes(s + Environment.NewLine)).ToArray();
///
///           // Compute a hash of the the byte array.
///           using (MD5CryptoServiceProvider hashProvider = new MD5CryptoServiceProvider())
///           {
///               this.Hash = hashProvider.ComputeHash(objectInBytes);
///           }
///           //Wrtie the Hash.
///           xmlWriter.WriteElementString("Hash", Convert.ToBase64String(this.Hash));
///
///           //IDEA: Could possibly enumerate through the types in UserCalibrationLimits to see if
type names and their underlying fields (with correct property names) exist and set them that way.
///           foreach (CalResult cr in this.Results)
///           {
///               xmlWriter.WriteStartElement(cr.ResultName);
///               foreach (ExpandoObject dynObj in cr.KvPMergedObjectData)
///               {
///                   xmlWriter.WriteStartElement($"KvPOf{cr.KeyName}{MaxMinDataTypeName}");
//$"KvPOf{cr.KeyName}{cr.ValueName}"
///                   xmlWriter.WriteStartElement("Key");
///                   foreach(var property in (IDictionary<String, Object>)dynObj)
///                   {
///                       var kvpRef = cr.KeyOrValuePropertyReferences.Find(x => x.Value ==
```

```
property.Key); //DOES NOT DESERIALIZE BACK IN FOR RE-USE!!!
///                     if (kvpRef.Key == "Key") //Only Add if it's a Key property, we do not need any
other Value properties.
///                     {
///                         if(property.Value != null)
///                         {
///                             xmlWriter.WriteElementString(property.Key, property.Value.ToString());
//TEMP edit: Inlined value string conditional.
///                         }
///                         else
///                         {
///                             xmlWriter.WriteStartElement(property.Key);
///                             xmlWriter.WriteAttributeString("xsi:nil", "true");
///                             xmlWriter.WriteEndElement();
///                         }
///                     }
///                 }
///                 xmlWriter.WriteEndElement(); //Ends Key element.
///                 xmlWriter.WriteStartElement("Value");
///                 foreach (var property in (IDictionary<String, Object>)dynObj)
///                 {
///                     switch (property.Key)
///                     {
///                         case "Min":
///                         case "MinTolerance":
///                         case "Max":
///                         case "MaxTolerance":
///                             xmlWriter.WriteElementString(property.Key, property.Value.ToString());
///                             break;
///                         case "CalFactor":
///                         case "ENR":
///                         case "Baseline":
///                             xmlWriter.WriteElementString("Baseline", property.Value.ToString());
//Name switches to "CalFactor" so MinMaxData can be de-serialized.
///                             break;
///                     }
///                 }
///                 xmlWriter.WriteEndElement(); //Ends Value element.
///                 xmlWriter.WriteEndElement(); //Ends KvPOf(KeyType+ValueType)
///             }
///             xmlWriter.WriteEndElement(); //Ends result name.
///         }
```

```
///
///            xmlWriter.WriteEndElement();
///            xmlWriter.WriteEndDocument();
///        }
///    }
///    catch (Exception ex)
///    {
///        //MessageBox.Show(ex.Message);
///        Console.WriteLine($"Calibration Limit File: {ex.Message}");
///    }
///}

/// <summary>
/// Gets every value (as object) of every property in every list of every result-Limit object. (And calibration data version property vlaue)
/// </summary>
/// <returns></returns>
private List<object> SavePropertiesToBeHashed()
{
List<object> objectList = new List<object>();
objectList.Add(this.CalibrationDataVersion.ToString());

foreach (CalResult result in this.Results) //May turn to LINQ
{
foreach (dynamic exObj in result.KvPMergedObjectData)
{
var dictionary = (IDictionary<string, object>)exObj;
foreach (var property in dictionary)
{
//string valueAsStr = property.Value != null ? property.Value.ToString() : ""; //TEMP edit: Inlined value string conditional.
objectList.Add(property.Value);
}
}
}
return objectList;
}

#endregion

public void Save()
{
```

```
CommitEditBeforeSave?.Invoke();
//this.Save_Custom_XML(this.FilePath);
this.SaveToXml(this.FilePath);
Console.WriteLine(string.Format("Saved (Write Hash = '{0}'): {1}",
Convert.ToBase64String(this.Hash), this.FilePath));
}
public void Load()
{

}
public bool Undo()
{
Console.WriteLine("Undo operation is not supported for Calibration Limits");
return true;
}

public bool Redo()
{
Console.WriteLine("Redo operation is not supported for Calibration Limits");
return true;
}

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
OnChangeOccured();
}
#endregion

public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}

}

}
```

CalPointModel.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Xml.Serialization;
using UnitConversionLib;

namespace MerlinTestStudio_Demo_Telerik.Data.Models
{
[Serializable]
public enum SourceOptions { SG1, SG2, NOISE };
[Serializable]
public enum SourcePortOptions { P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14,
P15, P16, HF_P1, HF_P2, HF_P3, HF_P4 };
[Serializable]
public enum MeasurePortOptions
{
P1,
P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16,
HF_P1, HF_P2, HF_P3, HF_P4,
M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15, M16,
HF_M1, HF_M2, HF_M3, HF_M4, HF_M5, HF_M6, HF_M7, HF_M8
};

//New, Add Support.
[Serializable]
public enum DoAttenCalEnum
{
[Description("Yes")]
Yes,
[Description("No")]
No,
[Description("Min Atten")]
Min_Atten
```

```
};

[TypeConverter(typeof(EnumDescriptionTypeConverter))]
[Serializable]
public enum MeasureFilterOptions
{
[Description("Atten")]
Atten,
[Description("Bypass")]
Bypass,
[Description ("HIGH") ]
FILT1,
[Description("LOW")]
FILT2,
[Description("MID")]
FILT3
};

[Serializable]
public enum PointValidationStatus
{
Valid = 0, //Default Color
Warning = 1, //Yellow
Invalid = 2 //Red
}


[Serializable]
public class CalPointModel : INotifyPropertyChanged
{
#region Private members
//Internal Reference Pins
private string _srcPinName = "";
private string _srcPinAlias = "";
private string _measPinName = "";
private string _measPinAlias = "";
private PinModel _srcPin;
private PinModel _measPin;
//CalPointV5 members
private CalType _calType;
private SourceOptions _source;
private string _sourcePort; //Not Displayed
```

```
private string _measurePort; //Not Displayed
private bool _useHfPath;
private double _frequency;
private double _deviceExpectedInputLevel;
private double _deviceExpectedOutputLevel;
private MeasureFilterOptions _measureFilter;
private double _measureAttenuation;
private bool _doAttenCal;
private AttenuationSettings _srcCalAttenSettings = AttenuationSettings.Crs_6dB_Fine_1_815V;
private RF110SrcAmpConfigurations? _LfSrcAmp;
private LfMeasureAmp? _LfMeasureAmp;
private HfAmp? _HfSrcAmp;
private HfAmp? _HfMeasAmp;
private LfAllAmp? _LfAllAmp;
private HfAllAmp? _HfAllAmp;
private string _Waveform;
private string _comment;
private bool _isVnaExtra;
//Validation
private PointValidationStatus _pointStatus;
private string _toolTipString = string.Empty;
//CalPoint Specific
private string _warningMessage = string.Empty;
private string _unit;
private string _sysPointVarient;
#endregion

#region Events
public event Action<CalPointModel> ValidationStatusChanged;
#endregion

#region Constructor

public CalPointModel()
{
//Empty constructor for deserialization.

CheckPointStatus();
}

/// <summary>
/// Constructor for creating an entirely new CalPointModel.
```

```
/// </summary>
/// <param name="unit"></param>
/// <param name="sysPointVarient"></param>
public CalPointModel(string unit, string sysPointVarient)
{
//Set Default Values
this._calType = CalType.RF;
this._source = SourceOptions.SG1;
this._sourcePort = "P1";
this._measurePort = "P2";
this._useHfPath = false;
this._frequency = UnitHelper.ValueUnitModifier(1e+9, "Hz", unit); //Default freq. 1 GHz converted
to the user selected unit.
this._deviceExpectedInputLevel = 0;
this._deviceExpectedOutputLevel = 0;
this._measureFilter = MeasureFilterOptions.Bypass;
this._measureAttenuation = 0;
this._doAttenCal = false;
this._Waveform = string.Empty;
this._comment = string.Empty;
this._isVnaExtra = false;

this._LfSrcAmp = null;
this._LfMeasureAmp = null;
this._HfSrcAmp = null;
this._HfMeasAmp = null;
this._LfAllAmp = null;
this._HfAllAmp = null;

this._unit = unit;
this._sysPointVarient = sysPointVarient;

CheckPointStatus();
}

/// <summary>
/// Constructor for creating an CalPointModel using imported CalPointV5 data.
/// </summary>
/// <param name="calPointBase"></param>
/// <param name="sysPointVarient"></param>
public CalPointModel(CalPointV5 calPointBase, string sysPointVarient, string unit)
{
```

```
this._calType = calPointBase.CalibrationType;
this._source = (SourceOptions)Enum.Parse(typeof(SourceOptions),
calPointBase.Source.ToString(), true);
this._sourcePort = calPointBase.SourcePort.ToString(); //
(SourcePortOptions)Enum.Parse(typeof(SourcePortOptions), calPointBase.SourcePort.ToString(),
true);
this._measurePort = calPointBase.MeasurePort.ToString(); //
(MeasurePortOptions)Enum.Parse(typeof(MeasurePortOptions),
calPointBase.MeasurePort.ToString(), true);
this._useHfPath = calPointBase.UseHfPath;
this._frequency = calPointBase.Frequency;
this._deviceExpectedInputLevel = calPointBase.DeviceExpectedInputLevel;
this._deviceExpectedOutputLevel = calPointBase.DeviceExpectedOutputLevel;
this._measureFilter = (MeasureFilterOptions)Enum.Parse(typeof(MeasureFilterOptions),
calPointBase.MeasureFilter.ToString(), true);
this._measureAttenuation = calPointBase.MeasureAttenuation;
this._doAttenCal = calPointBase.DoAttenCal;
this._Waveform = calPointBase.Waveform;
this._comment = calPointBase.Comment;
this._isVnaExtra = calPointBase.IsVnaExtra;

this._LfSrcAmp = calPointBase.Rf110LfSrcAmp;
this._LfMeasureAmp = calPointBase.Rf110LfMeasureAmp;
this._HfSrcAmp = calPointBase.Rf110HfSrcAmp;
this._HfMeasAmp = calPointBase.Rf110HfMeasAmp;
this._LfAllAmp = calPointBase.Rf210LfAllAmp;
this._HfAllAmp = calPointBase.Rf210HfAllAmp;

this._unit = "Hz";
this.Unit = unit;
this._sysPointVarient = sysPointVarient;

this.CheckPointStatus();
}
#endregion

#region Public Members
[JsonIgnore]
[XmlIgnore]
public int ID { get; set; } //Used in post processing
[JsonIgnore]
[XmlIgnore]
```

```
public int SiteNumber { get; set; } //Used in post processing
[JsonIgnore]
[XmlIgnore]
public bool IsVnaExtra //Used in post processing
{
get { return _isVnaExtra; }
set { _isVnaExtra = value; OnPropertyChanged("IsVnaExtra"); }
}

[JsonIgnore]
[XmlIgnore]
public PinModel SrcPin
{
get { return _srcPin; }
set
{
if (value != null)
{
string SrcIO = value.mappings.First().IO;
if (!string.IsNullOrEmpty(SrcIO))
{
_srcPin = value;
//value.PropertyChanged += this.PropertyChanged; //For name and pin alias
value.MappingChanged += Value_MappingChanged;
OnPropertyChanged("SrcPin");
this.SrcPinName = value.PinName;
this.SrcPinAlias = value.PinAlias;
}
else
{
Console.WriteLine("Src Pin provided an empty port, try a different pin or set port in Pin Map...");
}
}
else
{
_srcPin = value;
OnPropertyChanged("SrcPin");
//this.SrcPinName = "";
//this.SrcPinAlias = "";
}
}
}
```

```
[JsonIgnore]
[XmlIgnore]
public PinModel MeasPin
{
get { return _measPin; }
set
{
if (value != null)
{
string MeasIO = value.mappings.First().IO;
if (!string.IsNullOrEmpty(MeasIO))
{
_measPin = value;
//value.PropertyChanged += this.PropertyChanged; //For name and pin alias
value.MappingChanged += Value_MappingChanged;
OnPropertyChanged("MeasPin");
this.MeasPinName = value.PinName;
this.MeasPinAlias = value.PinAlias;
}
else
{
Console.WriteLine("Meas Pin provided an empty port, try a different pin or set port in Pin Map...");
}
}
else
{
_measPin = value;
OnPropertyChanged("MeasPin");
//this.MeasPinName = "";
//this.MeasPinAlias = "";
}
}
}

private void Value_MappingChanged(MappingModel obj)
{
CheckPointStatus();
//May need to set the Source/Measure Port, which ever it may be...
}

public CalType CalibrationType
{
```

```
get { return _calType; }
set { _calType = value; OnPropertyChanged("CalibrationType"); CheckPointStatus(); }
}
public SourceOptions Source
{
get { return _source; }
set { _source = value; OnPropertyChanged("Source"); CheckPointStatus(); }
}

public string SrcPinName
{
get
{
if (this.SrcPin != null)
{
return this.SrcPin.PinName;
}
else { return _srcPinName; }
}
set { _srcPinName = value; OnPropertyChanged("SrcPinName"); }
}
public string SrcPinAlias
{
get
{
if (this.SrcPin != null)
{
return this.SrcPin.PinAlias;
}
else { return _srcPinAlias; }
}
set
{
if (value != null)
{
_srcPinAlias = value;
}
else if (value is null)
{
_srcPinAlias = "";
}
OnPropertyChanged("SrcPinAlias");
```

```
}
}
public string MeasPinName
{
get
{
if (this.MeasPin != null)
{
return this.MeasPin.PinName;
}
else { return _measPinName; }
}
set { _measPinName = value; OnPropertyChanged("MeasPinName"); }
}
public string MeasPinAlias
{
get
{
if (this.MeasPin != null)
{
return this.MeasPin.PinAlias;
}
else { return _measPinAlias; }
}
set
{
if (value != null)
{
_measPinAlias = value;
}
else if (value is null)
{
_measPinAlias = "";
}
OnPropertyChanged("MeasPinAlias");
}
}
public bool UseHfPath
{
get { return _useHfPath; }
set { _useHfPath = value; OnPropertyChanged("UseHfPath"); CheckPointStatus(); }
}
```

```
public double Frequency
{
get { return _frequency; }
set { _frequency = value; OnPropertyChanged("Frequency"); CheckPointStatus(); }
}
public double DeviceExpectedInputLevel { get { return _deviceExpectedInputLevel; } set {
_deviceExpectedInputLevel = value; OnPropertyChanged("DeviceExpectedInputLevel");
CheckPointStatus(); } }
public double DeviceExpectedOutputLevel { get { return _deviceExpectedOutputLevel; } set {
_deviceExpectedOutputLevel = value; OnPropertyChanged("DeviceExpectedOutputLevel");
CheckPointStatus(); } }
public MeasureFilterOptions MeasureFilter { get { return _measureFilter; } set { _measureFilter =
value; OnPropertyChanged("MeasureFilter"); CheckPointStatus(); } }
public double MeasureAttenuation { get { return _measureAttenuation; } set { _measureAttenuation
= value; OnPropertyChanged("MeasureAttenuation"); CheckPointStatus(); } }
public bool DoAttenCal { get { return _doAttenCal; } set { _doAttenCal = value;
OnPropertyChanged("DoAttenCal"); CheckPointStatus(); } }
public AttenuationSettings SrcCalAttenSettings { get { return _srcCalAttenSettings; } set {
_srcCalAttenSettings = value; OnPropertyChanged("SrcCalAttenSettings"); CheckPointStatus(); }
}
public RF110SrcAmpConfigurations? Rf110LfSrcAmp
{
get { return _LfSrcAmp; }
set { _LfSrcAmp = value; OnPropertyChanged("Rf110LfSrcAmp"); CheckPointStatus(); }
}
public LfMeasureAmp? Rf110LfMeasureAmp
{
get { return _LfMeasureAmp; }
set { _LfMeasureAmp = value; OnPropertyChanged("Rf110LfMeasureAmp"); CheckPointStatus();
}
}
public HfAmp? Rf110HfSrcAmp
{
get { return _HfSrcAmp; }
set { _HfSrcAmp = value; OnPropertyChanged("Rf110HfSrcAmp"); CheckPointStatus(); }
}
public HfAmp? Rf110HfMeasAmp
{
get { return _HfMeasAmp; }
set { _HfMeasAmp = value; OnPropertyChanged("Rf110HfMeasAmp"); CheckPointStatus(); }
}
public LfAllAmp? Rf210LfAllAmp
```

```
{
get { return _LfAllAmp; }
set { _LfAllAmp = value; OnPropertyChanged("Rf210LfAllAmp"); CheckPointStatus(); }
}
public HfAllAmp? Rf210HfAllAmp
{
get { return _HfAllAmp; }
set { _HfAllAmp = value; OnPropertyChanged("Rf210HfAllAmp"); CheckPointStatus(); }
}

//private Waveform _waveformObject;
//[JsonIgnore]
//public Waveform WaveformObject
//{
//    get { return _waveformObject; }
//    set { _waveformObject = value; OnPropertyChanged("WaveformObject"); }
//}
public string Waveform
{
get { return _Waveform; }
set { _Waveform = value; OnPropertyChanged("Waveform"); }
}
public string Comment { get { return _comment; } set { _comment = value;
OnPropertyChanged("Comment"); } }

//GUI PROPERTIES
public PointValidationStatus PointStatus
{
get { return _pointStatus; }
set
{
_pointStatus = value;
OnPropertyChanged("PointStatus");
}
}
public string ToolTipString
{
get { return _toolTipString; }
set { _toolTipString = value; OnPropertyChanged("ToolTipString"); }
}
public bool HasWarning
{
```

```
get { return WarningMessage != string.Empty ? true : false; }
}
/// <summary>
/// The Warning Message used for certain Frequncy values.
/// </summary>
public string WarningMessage
{
get { return _warningMessage; }
set { _warningMessage = value; OnPropertyChanged("WarningMessage");
OnPropertyChanged("HasWarning"); }
}

/// <summary>
/// The Unit used for Frequncy value conversion.
/// </summary>
public string Unit
{
get { return _unit; }
set
{
string oldUnit = _unit;
_unit = value;
OnPropertyChanged("Unit");
Frequency = UnitHelper.ValueUnitModifier(Frequency, oldUnit, value);
}
}

/// <summary>
/// A reference to the system this CalPoint data will be consumed on. Used for changing data
validation rules.
/// Equivalent: UserTargetSystem
/// </summary>
public string SysPointVarient
{
get { return _sysPointVarient; }
set
{
_sysPointVarient = value;
OnPropertyChanged("SysPointVarient");
//Clears any prior warning message applicable only under a certain system.
//FIX, Doesn't work well, need better solution.
this.WarningMessage = string.Empty;
```

```
CheckPointStatus();
}
}
#endregion

#region Methods

[JsonIgnore]
[XmlIgnore]
public List<Guid> ErrorIDs = new List<Guid>();
/// <summary>
/// Checks the cal point validation status.
/// </summary>
private void CheckPointStatus()
{
List<ValidationResult> statusResults = new List<ValidationResult>();
double HzFreq = UnitHelper.ValueUnitModifier(this.Frequency, this.Unit, "Hz");

//Check CalType
switch (this.CalibrationType)
{
case CalType.Noise:
break;
case CalType.RF:
break;
case CalType.VNA_1P:
case CalType.VNA_2P:
if (this.Rf110LfSrcAmp == null || this.Rf110LfMeasureAmp == null || this.Rf210LfAllAmp == null)
{
ValidationResult VnaNullableEnumResult = new ValidationResult() { PropertyName = "VNA
CalType" };
VnaNullableEnumResult.ErrorMessage = string.Format(" The following cannot be null while VNA
is the selected CalType:" +
"\nRf110LfSrcAmp = {0},\nRf110LfMeasureAmp = {1},\nRf210LfAllAmp = {2}",
this.Rf110LfSrcAmp, this.Rf110LfMeasureAmp, this.Rf210LfAllAmp);
statusResults.Add(VnaNullableEnumResult);
}
break;
}

//Check Port Grouping on each site
if(this.SrcPin != null && this.MeasPin != null)
```

```
{
for (int i = 0; i < this.SrcPin.mappings.Count; i++)
{
int siteNum = i + 1;
string sourcePort = this.SrcPin.mappings[i].IO;
string measurePort = this.MeasPin.mappings[i].IO;

if (CalPointModel.Group(sourcePort) == CalPointModel.Group(measurePort))
{
ValidationResult SourcePortResult = new ValidationResult()
{
PropertyName = "Source / Measure Port",
ErrorMessage = string.Format("Cannot source and measure in the same grouping (Site {0}):\n -
Source Port: {1}\n - Measure Port: {2}", siteNum, sourcePort, measurePort)
};
statusResults.Add(SourcePortResult);
}
}
}

//Check UseHFPath
if (HzFreq < 6e+9 && this.UseHfPath == true)
{
ValidationResult UseHfPathResult = new ValidationResult()
{
PropertyName = "UseHfPath",
ErrorMessage = "Use HF Path requires a frequency greater than 6 GHz"
};
statusResults.Add(UseHfPathResult);
}

//Check Frequency
ValidationResult FrequencyResult = new ValidationResult() { PropertyName = "Frequency" };
switch (this.SysPointVarient)
{
case "APS-500":
switch (this.Source)
{
case SourceOptions.SG1:
if (HzFreq < 3.8e+8 || HzFreq > 1.2e+10)
{
FrequencyResult.ErrorMessage = "Frequency is out of range: SG1 - [380 MHz - 12 GHz]";
```

```
statusResults.Add(FrequencyResult);
}
if (HzFreq > 8e+9 && HzFreq <= 1.2e+10)
{
this.WarningMessage = "Warning: Option is required [8 GHz - 12 GHz]";
}
else { this.WarningMessage = string.Empty; }
break;
case SourceOptions.SG2:
if (HzFreq < 1e+8 || HzFreq > 2.15e+10)
{
FrequencyResult.ErrorMessage = "Frequency is out of range: SG2 - [100 MHz - 21.5 GHz]";
statusResults.Add(FrequencyResult);
}
else { this.WarningMessage = string.Empty; }
break;
case SourceOptions.NOISE:
if (HzFreq < 3.8e+8 || HzFreq > 1e+10)
{
FrequencyResult.ErrorMessage = "Frequency is out of range: NOISE - [380 MHz - 10 GHz]";
statusResults.Add(FrequencyResult);
}
else { this.WarningMessage = string.Empty; }
break;
}
break;
case "APS-300":
switch (this.Source)
{
case SourceOptions.SG1:
if (HzFreq < 2.5e+5 || HzFreq > 6e+9)
{
FrequencyResult.ErrorMessage = "Frequency is out of range: SG1 - [250 KHz - 6 GHz]";
statusResults.Add(FrequencyResult);
}
break;
case SourceOptions.SG2:
if (HzFreq < 1e+8 || HzFreq > 2.15e+10)
{
FrequencyResult.ErrorMessage = "Frequency is out of range: SG2 - [100 MHz - 21.5 GHz]";
statusResults.Add(FrequencyResult);
}
```

```
break;
case SourceOptions.NOISE:
if (HzFreq < 2.5e+5 || HzFreq > 6e+9)
{
FrequencyResult.ErrorMessage = "Frequency is out of range: NOISE - [250 KHz - 6 GHz]";
statusResults.Add(FrequencyResult);
}
break;
}
break;
case "APS-100":
switch (this.Source)
{
case SourceOptions.SG1:
break;
case SourceOptions.SG2:
break;
case SourceOptions.NOISE:
break;
}
break;
}

//Check DoAttenCal
ValidationResult DoAttenCalResult = new ValidationResult() { PropertyName = "DoAttenCal" };
switch (this.SysPointVarient)
{
case "APS-500":
switch (this.Source)
{
case SourceOptions.SG1:
case SourceOptions.SG2:
case SourceOptions.NOISE:
if (HzFreq > 8e+9 && this.DoAttenCal == true)
{
DoAttenCalResult.ErrorMessage = "Cannot Do Atten Cal with a Frequency > 8 GHz";
statusResults.Add(DoAttenCalResult);
}
break;
}
break;
case "APS-300":
```

```
switch (this.Source)
{
case SourceOptions.SG1:
//Doesn't allow a frequency above 6 GHz.
break;
case SourceOptions.SG2:
if (HzFreq > 6e+9 && this.DoAttenCal == true)
{
DoAttenCalResult.ErrorMessage = "Cannot Do Atten Cal with a frequency greater than 6 GHz";
statusResults.Add(DoAttenCalResult);
}
break;
case SourceOptions.NOISE:
//Doesn't allow a frequency above 6 GHz.
break;
}
break;
case "APS-100":
switch (this.Source)
{
case SourceOptions.SG1:
break;
case SourceOptions.SG2:
break;
case SourceOptions.NOISE:
break;
}
break;
}

//Error message handling logic goes here.
if(statusResults.Count() > 0)
{
PointStatus = PointValidationStatus.Invalid;
ToolTipString = string.Empty; //Clear ToolTip
foreach (ValidationResult vr in statusResults) //Re-compile ToolTip notifications.
{
if (!string.IsNullOrEmpty(ToolTipString)) { ToolTipString += Environment.NewLine; }
ToolTipString += string.Format("{0}: {1}.", vr.PropertyName, vr.ErrorMessage);
}
ValidationStatusChanged?.Invoke(this);
}
```

```
else
{
PointStatus = PointValidationStatus.Valid;
ToolTipString = string.Empty;
ValidationStatusChanged?.Invoke(this);
}

}

/// <summary>
/// Checks the source ports and returns an arbitrary group number.
/// </summary>
/// <param name="port"></param>
/// <returns></returns>
public static int Group(string port)
{
switch (port)
{
case "P1":
case "P9":
case "P13":
case "P5":
return 1;
case "P2":
case "P10":
case "P14":
case "P6":
return 2;
case "P3":
case "P11":
case "P15":
case "P7":
return 3;
case "P4":
case "P12":
case "P16":
case "P8":
return 4;
case "HF_P1":
return 10;
case "HF_P2":
return 20;
```

```
case "HF_P3":
return 30;
case "HF_P4":
return 40;
default: //All RF210 LF & HF ports.
return 0;
}
}

/// <summary>
/// Returns a CalPointV5 data object using GUI data from this CalPointModel.
/// </summary>
/// <returns></returns>
public CalPointV5 CreateCalPointV5()
{
double freq_in_Hz = UnitHelper.ValueUnitModifier(this.Frequency, this.Unit, "Hz");
CalPointV5 calPoint = new CalPointV5()
{
CalibrationType = this.CalibrationType,

SrcPinName = string.IsNullOrEmpty(this.SrcPinAlias) ? this.SrcPinName : this.SrcPinAlias,
SrcPinNameAlias = string.IsNullOrEmpty(this.SrcPinAlias) ? this.SrcPinAlias : this.SrcPinName,
MeasPinName = string.IsNullOrEmpty(this.MeasPinAlias) ? this.MeasPinName :
this.MeasPinAlias,
MeasPinNameAlias = string.IsNullOrEmpty(this.MeasPinAlias) ? this.MeasPinAlias :
this.MeasPinName,

Source = (SigGen)Enum.Parse(typeof(SigGen), this.Source.ToString(), true),
//SourcePort = (SrcPort)Enum.Parse(typeof(SrcPort), this.SourcePort.ToString(), true),
//MeasurePort = (MeasPort)Enum.Parse(typeof(MeasPort), this.MeasurePort.ToString(), true),
UseHfPath = this.UseHfPath,
Frequency = freq_in_Hz,
DeviceExpectedInputLevel = this.DeviceExpectedInputLevel,
DeviceExpectedOutputLevel = this.DeviceExpectedOutputLevel,
MeasureFilter = (MeasFilt)Enum.Parse(typeof(MeasFilt), this.MeasureFilter.ToString(), true),
MeasureAttenuation = this.MeasureAttenuation,
DoAttenCal = this.DoAttenCal,
SrcCalAttenSettings = this.SrcCalAttenSettings,
Rf110LfSrcAmp = this.Rf110LfSrcAmp,
Rf110LfMeasureAmp = this.Rf110LfMeasureAmp,
Rf110HfSrcAmp = this.Rf110HfSrcAmp,
Rf110HfMeasAmp = this.Rf110HfMeasAmp,
```

```
Rf210LfAllAmp = this.Rf210LfAllAmp,
Rf210HfAllAmp = this.Rf210HfAllAmp,
Waveform = this.Waveform,
Comment = this.Comment,
IsVnaExtra = this.IsVnaExtra
};
return calPoint;
}

public static CalPointV5 CloneCalPointV5(CalPointV5 calPointV5)
{
CalPointV5 clone = new CalPointV5()
{
CalibrationType = calPointV5.CalibrationType,
SrcPinName = calPointV5.SrcPinName,
SrcPinNameAlias = calPointV5.SrcPinNameAlias,
MeasPinName = calPointV5.MeasPinName,
MeasPinNameAlias = calPointV5.MeasPinNameAlias,
Source = calPointV5.Source,
SourcePort = calPointV5.SourcePort,
MeasurePort = calPointV5.MeasurePort,
UseHfPath = calPointV5.UseHfPath,
Frequency = calPointV5.Frequency,
DeviceExpectedInputLevel = calPointV5.DeviceExpectedInputLevel,
DeviceExpectedOutputLevel = calPointV5.DeviceExpectedOutputLevel,
MeasureFilter = calPointV5.MeasureFilter,
MeasureAttenuation = calPointV5.MeasureAttenuation,
DoAttenCal = calPointV5.DoAttenCal,
SrcCalAttenSettings = calPointV5.SrcCalAttenSettings,
Rf110LfSrcAmp = calPointV5.Rf110LfSrcAmp,
Rf110LfMeasureAmp = calPointV5.Rf110LfMeasureAmp,
Rf110HfSrcAmp = calPointV5.Rf110HfSrcAmp,
Rf110HfMeasAmp = calPointV5.Rf110HfMeasAmp,
Rf210LfAllAmp = calPointV5.Rf210LfAllAmp,
Rf210HfAllAmp = calPointV5.Rf210HfAllAmp,
Waveform = calPointV5.Waveform,
Comment = calPointV5.Comment,
ID = calPointV5.ID,
IsVnaExtra = calPointV5.IsVnaExtra
};
return clone;
}
```

```
public CalPointModel GetClone()
{
CalPointModel clone = new CalPointModel(Unit, SysPointVarient)
{
CalibrationType = this.CalibrationType,
Source = this.Source,
SrcPin = this.SrcPin,
MeasPin = this.MeasPin,
SrcPinName = this.SrcPinName,
SrcPinAlias = this.SrcPinAlias,
MeasPinName = this.MeasPinName,
MeasPinAlias = this.MeasPinAlias,
//SourcePort = this.SourcePort,
//MeasurePort = this.MeasurePort,
UseHfPath = this.UseHfPath,
Frequency = this.Frequency,
DeviceExpectedInputLevel = this.DeviceExpectedInputLevel,
DeviceExpectedOutputLevel = this.DeviceExpectedOutputLevel,
MeasureFilter = this.MeasureFilter,
MeasureAttenuation = this.MeasureAttenuation,
DoAttenCal = this.DoAttenCal,
SrcCalAttenSettings = this.SrcCalAttenSettings,
Rf110LfSrcAmp = this.Rf110LfSrcAmp,
Rf110LfMeasureAmp = this.Rf110LfMeasureAmp,
Rf110HfSrcAmp = this.Rf110HfSrcAmp,
Rf110HfMeasAmp = this.Rf110HfMeasAmp,
Rf210LfAllAmp = this.Rf210LfAllAmp,
Rf210HfAllAmp = this.Rf210HfAllAmp,
Waveform = this.Waveform,
Comment = this.Comment,

IsVnaExtra = this.IsVnaExtra,

Unit = this.Unit,
SysPointVarient = this.SysPointVarient,
WarningMessage = this.WarningMessage,
};
return clone;
}

#endregion
```

```
#region INPC
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}

//This class is only used in the CheckPointStatus() method to keep track of errors internally and is not used externally.
[Serializable]
public class ValidationResult
{
public int ID { get; set; }
public string PropertyName { get; set; }
public string ErrorMessage { get; set; }

}
}
```

CalResult.cs

```
using MerlinTest.Common.Types;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Models
{
public interface ICalResult
{
string CalDataModelParent { get; set; }//Eliminate usage.
string ResultName { get; set; }
List<ICalFactorKey> CalFactorKeys { get; set; }
List<object> KvPMergedObjectData { get; set; }
}
```

```
[Serializable]
public class CalResult : ICalResult, INotifyPropertyChanged
{
public CalDataModel Parent { get; set; }

public string CalDataModelParent { get; set; } //Eliminate usage.

public string ResultName { get; set; }
public List<ICalFactorKey> CalFactorKeys { get; set; }

public string KeyName { get; set; }
public string ValueName { get; set; }
//[KvP<KeyOrValue, PropertyName>]
public List<KvP<string, string>> KeyOrValuePropertyReferences { get; set; } //Used for tracking
which properties are key or value in KvPMergedObjectData list.
public List<dynamic> KvPMergedObjectData { get; set; }

#region INPC
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
public void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
}
```

HistoryRecord.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Models
{
public struct Edit<I, V, P>
{
public Edit(I i, V v, P p) : this()
{
```

```
Index = i;
Value = v;
Property = p;
}
public I Index { get; set; }
public V Value { get; set; }
public P Property { get; set; }
}

[Serializable]
public class HistoryRecord
{
/// <summary>
/// The original index of the item in the collection
/// </summary>
public int Index { get; set; }

public object DataItem { get; set; }

/// <summary>
/// The name of the property which has been changed
/// </summary>
public string Property { get; set; }

/// <summary>
/// Original value of the property
/// </summary>
public object Value { get; set; }

/// <summary>
/// The action that has been performed
/// </summary>
public HistoryAction Action { get; set; }
}

public enum HistoryAction
{
Delete,
Add,
Duplicate,
Edit
}
```

```
}
```

DigitalLevelsModel.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Xml;
using System.Xml.Linq;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels
{
//Maybe try using a Serializable abstract type for levels.
#region Levels Types
[Serializable]
public class DigiLevel
{
private PinModel _digiPin = null;
[XmlIgnore]
public PinModel DigiPin
{
get { return _digiPin; }
set
{
_digiPin = value; OnPropertyChanged("DigiPin");
if (value != null)
{
PinItem = value.PinName;
}
}
}
public string PinItem { get; set; } //PinName
public double? VIH { get; set; }
public double? VIL { get; set; }
```

```
public double? VOH { get; set; }
public double? VOL { get; set; }
public string TerminationMode { get; set; }
public double? VTERM { get; set; }
public double? VCOM { get; set; }
public double? IOL { get; set; }
public double? IOH { get; set; }
public string Comment { get; set; }
#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
#endregion
}
[Serializable]
public class PPMULevel
{
private PinModel _digiPin = null;
[XmlIgnore]
public PinModel DigiPin
{
get { return _digiPin; }
set
{
_digiPin = value; OnPropertyChanged("DigiPin");
if (value != null)
{
PinItem = value.PinName;
}
}
}
public string PinItem { get; set; } //PinName
public double? VF { get; set; }
public double? IF { get; set; }
public double? VCH { get; set; }
public double? VCL { get; set; }
public double? IRange { get; set; }
public string Comment { get; set; }
#region INPC Members
```

```
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
#endregion
}

[Serializable]
public class DCPowerLevel
{
private PinModel _digiPin = null;
private string _pinItem;

[XmlIgnore]
public PinModel DigiPin
{
get { return _digiPin; }
set
{
_digiPin = value; OnPropertyChanged("DigiPin");
if(value != null)
{
PinItem = value.PinName;
}
}
}
public string PinItem //PinName
{
get { return _pinItem; }
set { _pinItem = value; OnPropertyChanged("PinItem"); }
}
public double? VF { get; set; }
public double? IC { get; set; }
public double? IF { get; set; }
public double? VC { get; set; }
public double? IRange { get; set; }
public double? VRange { get; set; }
public string Sense { get; set; }
public string Comment { get; set; }
#region INPC Members
```

```
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
#endregion
}
#endregion

[Serializable]
//[XmlRoot("DigitalLevelsModel")]
public class  DigitalLevelsModel : INotifyPropertyChanged, IFileData
{
#region Private Members
private string _filePath;
private ObservableCollection<DigiLevel> _digiLevelsSource = new
ObservableCollection<DigiLevel>();
private ObservableCollection<PPMULevel> _pPMULevelsSource = new
ObservableCollection<PPMULevel>();
private ObservableCollection<DCPowerLevel> _dCPowerLevelsSource = new
ObservableCollection<DCPowerLevel>();
#endregion

#region Public Members
public string FileName => Path.GetFileName(this.FilePath);
[XmlIgnore]
public string FilePath
{
get { return _filePath; }
set { _filePath = value; OnPropertyChanged("FilePath"); OnPropertyChanged("FileName"); }
}
public ObservableCollection<DigiLevel> DigiLevelsSource
{
get { return _digiLevelsSource; }
set { _digiLevelsSource = value; OnPropertyChanged("DigiLevelsSource"); }
}
public ObservableCollection<PPMULevel> PPMULevelsSource
{
get { return _pPMULevelsSource; }
set { _pPMULevelsSource = value; OnPropertyChanged("PPMULevelsSource"); }
}
```

```
public ObservableCollection<DCPowerLevel> DCPowerLevelsSource
{
get { return _dCPowerLevelsSource; }
set { _dCPowerLevelsSource = value; OnPropertyChanged("DCPowerLevelsSource"); }
}
#endregion

#region Events
public event Action ChangeOccured;
public event Action CommitEditBeforeSave;
#endregion

#region Constructor
public DigitalLevelsModel() { } //Deserialization Constructor
public DigitalLevelsModel(string filePath)
{
this.FilePath = filePath;
}
#endregion

public void Save()
{
CommitEditBeforeSave?.Invoke();
this.Save_NI_XML(this.FilePath);
Console.WriteLine(string.Format("Saved: {0}", this.FilePath));
}
public void Load()
{

}
public bool Undo()
{
Console.WriteLine("Undo operation not implemented for Digital Level Files");
return true;
}

public bool Redo()
{
Console.WriteLine("Redo operation not implemented for Digital Level Files");
return true;
}
```

```
#region XML Import/Export

//Helper Method
private bool IsCollectionItemsHasValues(IEnumerable<object>  collection)
{
for (int i = 0; i < collection.Count(); i++)
{
if (IsAnyValueStored(collection.ElementAt(i)))
{
return true;
}
}
return false;
}
//Helper Method
private bool IsAnyValueStored(object myObject)
{
foreach (PropertyInfo pi in myObject.GetType().GetProperties())
{
if (pi.PropertyType == typeof(double?))
{
double? value = (double?)pi.GetValue(myObject);
if (value != null)
{
return true;
}
}
else if (pi.PropertyType == typeof(string))
{
string value = (string)pi.GetValue(myObject);
if (!string.IsNullOrEmpty(value))
{
return true;
}
}
else if (pi.PropertyType == typeof(PinModel))
{
PinModel value = (PinModel)pi.GetValue(myObject);
if (value != null)
{
return true;
}
```

```
}
}
return false;
}

public void Save_NI_XML(string FileName)
{
try
{
//string NiFilePath = Path.Combine(Path.GetDirectoryName(FileName),
$"{Path.GetFileNameWithoutExtension(FileName)} (Compile){Path.GetExtension(FileName)}");
using (XmlTextWriter xmlWriter = new XmlTextWriter(FileName, System.Text.Encoding.UTF8) {
Formatting = Formatting.Indented })
{
xmlWriter.WriteStartDocument();
xmlWriter.WriteStartElement("PinLevelsFile");
xmlWriter.WriteAttributeString("xmlns:xsd", "http://www.w3.org/2001/XMLSchema");
xmlWriter.WriteAttributeString("xmlns:xsi", "http://www.w3.org/2001/XMLSchema-instance");
xmlWriter.WriteAttributeString("schemaVersion", "1.0");
xmlWriter.WriteAttributeString("xmlns", "http://www.ni.com/Semiconductor/PinLevels");

xmlWriter.WriteStartElement("PinLevelsSheet");
if(IsCollectionItemsHasValues(this.DigiLevelsSource) == true)
{
xmlWriter.WriteStartElement("DigitalPinLevelSets");
foreach (var digiLevel in this.DigiLevelsSource)
{
if (digiLevel.DigiPin != null)
{
xmlWriter.WriteStartElement("DigitalPinLevelSet");
xmlWriter.WriteAttributeString("pin", digiLevel.PinItem);
if (digiLevel.VIL != null) { xmlWriter.WriteElementString("Vil", digiLevel.VIL.ToString()); }
if (digiLevel.VIH != null) { xmlWriter.WriteElementString("Vih", digiLevel.VIH.ToString()); }
if (digiLevel.VOL != null) { xmlWriter.WriteElementString("Vol", digiLevel.VOL.ToString()); }
if (digiLevel.VOH != null) { xmlWriter.WriteElementString("Voh", digiLevel.VOH.ToString()); }
if (digiLevel.IOL != null) { xmlWriter.WriteElementString("Iol", digiLevel.IOL.ToString()); }
if (digiLevel.IOH != null) { xmlWriter.WriteElementString("Ioh", digiLevel.IOH.ToString()); }
if (digiLevel.VCOM != null) { xmlWriter.WriteElementString("Vcom", digiLevel.VCOM.ToString()); }
if (digiLevel.VTERM != null) { xmlWriter.WriteElementString("Vterm", digiLevel.VTERM.ToString());
}
if (!string.IsNullOrEmpty(digiLevel.TerminationMode)) {
xmlWriter.WriteElementString("TerminationMode", digiLevel.TerminationMode); }
```

```
if (!string.IsNullOrEmpty(digiLevel.Comment)) { xmlWriter.WriteElementString("Comment",
digiLevel.Comment); }
xmlWriter.WriteEndElement();
}
//else { Console.WriteLine($"{FileName}: Cannot serialize a DigitalPinLevelSet that has a null Pin
Item."); }
}
xmlWriter.WriteEndElement();
}
if (IsCollectionItemsHasValues(this.PPMULevelsSource) == true)
{
xmlWriter.WriteStartElement("PpmuPinLevelSets");
foreach (var ppmuLevel in this.PPMULevelsSource)
{
if (ppmuLevel.DigiPin != null)
{
xmlWriter.WriteStartElement("PpmuPinLevelSet");
xmlWriter.WriteAttributeString("pin", ppmuLevel.PinItem);
if (ppmuLevel.VF != null) { xmlWriter.WriteElementString("Vf", ppmuLevel.VF.ToString()); }
if (ppmuLevel.IF != null) { xmlWriter.WriteElementString("If", ppmuLevel.IF.ToString()); }
if (ppmuLevel.VCH != null) { xmlWriter.WriteElementString("Vch", ppmuLevel.VCH.ToString()); }
if (ppmuLevel.VCL != null) { xmlWriter.WriteElementString("Vcl", ppmuLevel.VCL.ToString()); }
if (ppmuLevel.IRange != null) { xmlWriter.WriteElementString("IRange",
ppmuLevel.IRange.ToString()); }
if (!string.IsNullOrEmpty(ppmuLevel.Comment)) { xmlWriter.WriteElementString("Comment",
ppmuLevel.Comment); }
xmlWriter.WriteEndElement();
}
//else { Console.WriteLine($"{FileName}: Cannot serialize a PpmuPinLevelSet that has a null Pin
Item."); }
}
xmlWriter.WriteEndElement();
}
if (IsCollectionItemsHasValues(this.DCPowerLevelsSource) == true)
{
xmlWriter.WriteStartElement("PowerPinLevelSets");
foreach (var powerLevel in this.DCPowerLevelsSource)
{
if (powerLevel.DigiPin != null)
{
xmlWriter.WriteStartElement("PowerPinLevelSet");
xmlWriter.WriteAttributeString("pin", powerLevel.PinItem);
```

```
if (powerLevel.VF != null) { xmlWriter.WriteElementString("Vf", powerLevel.VF.ToString()); }
if (powerLevel.IC != null) { xmlWriter.WriteElementString("Ic", powerLevel.IC.ToString()); }
if (powerLevel.IF != null) { xmlWriter.WriteElementString("If", powerLevel.IF.ToString()); }
if (powerLevel.VC != null) { xmlWriter.WriteElementString("Vc", powerLevel.VC.ToString()); }
if (powerLevel.IRange != null) { xmlWriter.WriteElementString("IRange",
powerLevel.IRange.ToString()); }
if (powerLevel.VRange != null) { xmlWriter.WriteElementString("VRange",
powerLevel.VRange.ToString()); }
if (!string.IsNullOrEmpty(powerLevel.Sense)) { xmlWriter.WriteElementString("Sense",
powerLevel.Sense); }
if (!string.IsNullOrEmpty(powerLevel.Comment)) { xmlWriter.WriteElementString("Comment",
powerLevel.Comment); }
xmlWriter.WriteEndElement();
}
//else { Console.WriteLine($"{FileName}: Cannot serialize a PowerPinLevelSet that has a null Pin
Item."); }
}
xmlWriter.WriteEndElement();
}
xmlWriter.WriteEndElement();

xmlWriter.WriteEndElement();
xmlWriter.WriteEndDocument();
}
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
}
public static DigitalLevelsModel Load_NI_Xml(string FileName)
{
DigitalLevelsModel LevelsModel = new DigitalLevelsModel() { FilePath = FileName };

XDocument document = XDocument.Load(FileName);

foreach (XElement pinLevelSheet in document.Root.Descendants())
{
foreach (XElement sectionSet in pinLevelSheet.Descendants())
{
switch (sectionSet.Name.LocalName)
```

```
{
case "DigitalPinLevelSets":
foreach (XElement pinLevelSet in sectionSet.Descendants())
{
if (pinLevelSet.Name.LocalName == "DigitalPinLevelSet")
{
var digiPinSet = new DigiLevel();
digiPinSet.PinItem = pinLevelSet.Attributes().FirstOrDefault().Value;
foreach (XElement property in pinLevelSet.Descendants())
{
switch (property.Name.LocalName)
{
case "Vil": digiPinSet.VIL = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "Vih": digiPinSet.VIH = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "Vol": digiPinSet.VOL = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "Voh": digiPinSet.VOH = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "Iol": digiPinSet.IOL = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "Ioh": digiPinSet.IOH = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "Vcom": digiPinSet.VCOM = double.Parse(pinLevelSet.Element(property.Name).Value);
break;
case "Vterm": digiPinSet.VTERM = double.Parse(pinLevelSet.Element(property.Name).Value);
break;
case "TerminationMode": digiPinSet.TerminationMode =
pinLevelSet.Element(property.Name).Value; break;
case "Comment": digiPinSet.Comment = pinLevelSet.Element(property.Name).Value; break;
}
}
LevelsModel.DigiLevelsSource.Add(digiPinSet);
}
}
break;
case "PpmuPinLevelSets":
foreach (XElement pinLevelSet in sectionSet.Descendants())
{
if (pinLevelSet.Name.LocalName == "PpmuPinLevelSet")
{
var ppmuPinSet = new PPMULevel();
ppmuPinSet.PinItem = pinLevelSet.Attributes().FirstOrDefault().Value;
foreach (XElement property in pinLevelSet.Descendants())
{
switch (property.Name.LocalName)
{
```

```
case "Vf": ppmuPinSet.VF = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "If": ppmuPinSet.IF = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "Vch": ppmuPinSet.VCH = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "Vcl": ppmuPinSet.VCL = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "IRange": ppmuPinSet.IRange = double.Parse(pinLevelSet.Element(property.Name).Value);
break;
case "Comment": ppmuPinSet.Comment = pinLevelSet.Element(property.Name).Value; break;
}
}
LevelsModel.PPMULevelsSource.Add(ppmuPinSet);
}
}
break;
case "PowerPinLevelSets":
foreach (XElement pinLevelSet in sectionSet.Descendants())
{
if (pinLevelSet.Name.LocalName == "PowerPinLevelSet")
{
var powPinSet = new DCPowerLevel();
powPinSet.PinItem = pinLevelSet.Attributes().FirstOrDefault().Value;
foreach (XElement property in pinLevelSet.Descendants())
{
switch (property.Name.LocalName)
{
case "Vf": powPinSet.VF = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "Ic": powPinSet.IC = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "If": powPinSet.IF = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "Vc": powPinSet.VC = double.Parse(pinLevelSet.Element(property.Name).Value); break;
case "IRange": powPinSet.IRange = double.Parse(pinLevelSet.Element(property.Name).Value);
break;
case "VRange": powPinSet.VRange = double.Parse(pinLevelSet.Element(property.Name).Value);
break;
case "Sense": powPinSet.Sense = pinLevelSet.Element(property.Name).Value; break;
case "Comment": powPinSet.Comment = pinLevelSet.Element(property.Name).Value; break;
}
}
LevelsModel.DCPowerLevelsSource.Add(powPinSet);
}
}
break;
}
```

```
}
}

return LevelsModel;
}
#endregion

public void RelinkPinData(PinMapModel projectPinMap)
{
foreach (var Level in this.DigiLevelsSource)
{
foreach (Data.PinModel pin in projectPinMap.DigitalPins.Where(pin => pin.PinName ==
Level.PinItem))
{
Level.DigiPin = pin;
}
}
foreach (var Level in this.PPMULevelsSource)
{
foreach (Data.PinModel pin in projectPinMap.DigitalPins.Where(pin => pin.PinName ==
Level.PinItem))
{
Level.DigiPin = pin;
}
}
foreach (var Level in this.DCPowerLevelsSource)
{
foreach (Data.PinModel pin in projectPinMap.DigitalPins.Where(pin => pin.PinName ==
Level.PinItem))
{
Level.DigiPin = pin;
}
}
}

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
```

```
OnChangeOccured();
}
#endregion

public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
}
}
```

DigitalPatternModel.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels
{
//public class OpcodeTag
//{
//
//}

[Serializable]
public class Mode : INotifyPropertyChanged
{
#region Private Members
private string _name = string.Empty;
private List<string> _registers = new List<string>();
private string _dPatSrcData = string.Empty;
private ObservableCollection<ModeVector> _modeVectors = new
ObservableCollection<ModeVector>();
```

```
#endregion

#region Public Members
public string Name
{
get { return _name; }
set { _name = value; OnPropertyChanged("Name"); }
}

//OLD here for backwards compatibility.
public List<string> RegData //Data For Register. //REPLACED BY ModeVectors
{
get { return _registers; }
set { _registers = value; OnPropertyChanged("RegData"); }
}

public ObservableCollection<ModeVector> ModeVectors
{
get { return _modeVectors; }
set { _modeVectors = value; OnPropertyChanged("ModeVectors"); }
}

[XmlIgnore]
public bool IsExpandable => OpcodesDetected.Count() > 0 ? true : false;

private ObservableCollection<Operation> _opcodesDetected = new
ObservableCollection<Operation>();
[XmlIgnore]
public ObservableCollection<Operation> OpcodesDetected
{
get { return _opcodesDetected; }
set { _opcodesDetected = value; OnPropertyChanged("OpcodesDetected"); }
}

[XmlIgnore]
public string DPatSrcData
{
get { return _dPatSrcData; }
set { _dPatSrcData = value; OnPropertyChanged("DPatSrcData"); }
}

#endregion
```

```
#region Constructor
public Mode()
{

}
#endregion

public void FindDetectedOpcodes()
{
this.OpcodesDetected.Clear();

foreach (ModeVector mv in this.ModeVectors)
{
foreach (Operation op in mv.VectorizedData)
{
if (!string.IsNullOrEmpty(op.Opcode))
{
OpcodesDetected.Add(op);
}
}
}

OnPropertyChanged("IsExpandable");
}

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
#endregion
}

[Serializable]
public class ModeVector : INotifyPropertyChanged
{
private string _inputValue = "0x00";
private ObservableCollection<Operation> _vectorizedData = new
ObservableCollection<Operation>();
```

```
public string InputValue
{
get { return _inputValue; }
set { _inputValue = value; OnPropertyChanged("InputValue");
}
}

//Calculated on import and edits.
public ObservableCollection<Operation> VectorizedData
{
get { return _vectorizedData; }
set { _vectorizedData = value; OnPropertyChanged("VectorizedData"); }
}

#region Constructors
public ModeVector() { } //Deserialization Constructor.
public ModeVector(string inputValue)
{
this.InputValue = inputValue;
}
#endregion

#region Methods
public ObservableCollection<Operation> VectorizeInput(string registerAddress, string timeSet)
{
List<Operation> MasterOps = new List<Operation>();

string data = this.InputValue;
string register = Regex.Replace(registerAddress, "_", "", RegexOptions.Compiled); //[^a - zA - Z0 -
9_.]+

#region Pre-Process Data Checks
//Break if the data is not required. Should break if the data string does not contain an x or perhaps
a 0x : TBD.
if (data == "no need" || data == "No Need" || data == "NN" || data == "nn" ||
string.IsNullOrEmpty(data))
{
return new ObservableCollection<Operation>(MasterOps);
}
//For some reason it allows the letter "a" through and breaks the entire process.
if (!DigitalPatternModel.OnlyHexInString(data)) //Checks if the data only contains only C-style hex
```

```
notation (0xFF) values
{
Console.WriteLine($"RFFE input invalid: {data}, invalid hex data for Register {register}");
return new ObservableCollection<Operation>(MasterOps);
}
#endregion Pre-Process Data Checks

// Convert the register and data values from hex strings to integers
int registerInt = Convert.ToInt32(register, 16);
int dataInt = Convert.ToInt32(data, 16);

// Convert the register and data values to binary strings
string registerBinary = Convert.ToString(registerInt, 2).PadLeft(8, '0');
string dataBinary = Convert.ToString(dataInt, 2).PadLeft(8, '0');

// Create a sequence of operations for the start sequence
List<Operation> startSequence = new List<Operation>()
{
new Operation(timeSet, 0.ToString(), "X") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data }, //REPEAT(3)
new Operation(timeSet, 0.ToString(), "X") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "X") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },

new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data }, //REPEAT(6)
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },

new Operation(timeSet, 0.ToString(), "1") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data }
```

```
};

// Create a sequence of operations for the command
string initialCommandSectionDataLineValues = "1111 010 "; //As seen below. (Non-dynamic initial sequence)
List<Operation> command = new List<Operation>() //(Non - dynamic initial sequence)
{
new Operation(timeSet, 1.ToString(), "1") { Section = VectorSection.Command, Comment = "//SA3", Original_Input = data }, //REPEAT(4) //Slave Address
new Operation(timeSet, 1.ToString(), "1") { Section = VectorSection.Command, Comment = "//SA2", Original_Input = data },
new Operation(timeSet, 1.ToString(), "1") { Section = VectorSection.Command, Comment = "//SA1", Original_Input = data },
new Operation(timeSet, 1.ToString(), "1") { Section = VectorSection.Command, Comment = "//SA0", Original_Input = data },

new Operation(timeSet, 1.ToString(), "0") { Comment = "//C7", Section = VectorSection.Command, Original_Input = data }, //C7 //Sets to Write //(0 1 0: Write --- 0 1 1: Read)
new Operation(timeSet, 1.ToString(), "1") { Comment = "//C6", Section = VectorSection.Command, Original_Input = data }, //C6 //Sets to Write
new Operation(timeSet, 1.ToString(), "0") { Comment = "//C5", Section = VectorSection.Command, Original_Input = data }, //C5 //Sets to Write
};
for (int i = 3; i < 8; i++) //reg address: C4, C3, C2, C1, C0
{
command.Add(new Operation(timeSet, 1.ToString(), registerBinary[i].ToString()) { Comment = "//C" + (registerBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input = data });
}
//Append slave address bits and write/read bits (first 3 C7,6,5 bits) then Cut off the first 3 bits of registerBinary before calculating the parity bit
registerBinary = initialCommandSectionDataLineValues + registerBinary.Substring(3);
int commandParityBit = registerBinary.Count(c => c == '1') % 2; //command
string commandParityBitString = commandParityBit == 0 ? "1" : "0"; //command
command.Add(new Operation(timeSet, 1.ToString(), commandParityBitString) { Comment = "//P", Section = VectorSection.Command, Original_Input = data }); //Add in the parity bit.

// Create a sequence of operations for the data
List<Operation> dataSequence = new List<Operation>();
for (int i = 0; i < 8; i++) //data: D7, D6, D5, D4, D3, D2, D1, D0
{
dataSequence.Add(new Operation(timeSet, 1.ToString(), dataBinary[i].ToString()) { Comment =
```

```
"//D" + (dataBinary.Length - (i + 1)), Section = VectorSection.Data, Original_Input = data });
}
// Calculate the parity bit for the data and command
int dataParityBit = dataBinary.Count(c => c == '1') % 2; //data
string dataParityBitString = dataParityBit == 0 ? "1" : "0"; //data
dataSequence.Add(new Operation(timeSet, 1.ToString(), dataParityBitString) { Comment = "//P",
Section = VectorSection.Data, Original_Input = data }); //Write in the parity bit.

// Create a sequence of operations for the bus park
List<Operation> busPark = new List<Operation>()
{
new Operation(timeSet, 1.ToString(), "0") { Section = VectorSection.Bus_Park, Comment =
"//BPC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "X") { Section = VectorSection.Bus_Park, Comment =
"//BPC", Original_Input = data },
new Operation(timeSet, "0", "0") { Section = VectorSection.Bus_Park, Comment = "//BPC",
Original_Input = data }
};

//comine all operations to master list!
MasterOps.AddRange(startSequence);
MasterOps.AddRange(command);
MasterOps.AddRange(dataSequence);
MasterOps.AddRange(busPark);

//Apply vector numbers to all the operations in master ops.
for (int i = 0; i < MasterOps.Count; i++)
{
MasterOps[i].VectorNumber = i + 1;
}

return new ObservableCollection<Operation>(MasterOps);
}

//NEEDS correct placement for Pattern Spec Determination during editing and compiling so
determination is real time and dynamic.
public static RffePatternSpec FindRffePatternSpec(string input)
{
foreach (var kvp in BackendConstants.RffePatternSpecKvPs)
{
if (input.Contains(kvp.Value))
{
```

```
return kvp.Key;
}
}

return RffePatternSpec.W; //Return Default (if not found in the string).
//throw new ArgumentException("No matching RffePatternSpec found.");
}

//For removing the RffePatternSpec characters if it's included in the data input.
public static string RemoveNonHexCharacters(string input)
{
// Use a regular expression to match non-hex characters
var regex = new Regex("[^0-9a-fA-F]+");

// Remove non-hex characters and return the result
return regex.Replace(input, "");
}
#endregion

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
#endregion
}

//Opcodes enum
//[Serializable]
//public enum VectorOpcodes
//{
//    [Description("SET_FLAGS(CPU_A1)")]
//    SET_FLAGS_CPU_A1,
//    [Description("SET_FLAGS(CPU_A2)")]
//    SET_FLAGS_CPU_A2,
//    [Description("SET_FLAGS(CPU_B1)")]
//    SET_FLAGS_CPU_B1,
//    [Description("SET_FLAGS(CPU_B2)")]
//    SET_FLAGS_CPU_B2,
//    [Description("SET_FLAGS(CPU_C1)")]
```

```
//    SET_FLAGS_CPU_C1,
//    [Description("SET_FLAGS(CPU_C2)")]
//    SET_FLAGS_CPU_C2,
//    [Description("SET_FLAGS(CPU_D1)")]
//    SET_FLAGS_CPU_D1,
//    [Description("SET_FLAGS(CPU_D2)")]
//    SET_FLAGS_CPU_D2,
//}
//NO REPEAT OR HALT FOR USER OPTIONS

[Serializable]
public class DigitalPatternModel : INotifyPropertyChanged, IFileData
{
#region Private Members
private string _filePath;
private DigiTimeObj _timeSetUsedObj;
private string _timeSetItem;
private PinModel _clockPin;
private string _clockPinItem;
private PinModel _dataPin;
private string _dataPinItem;
private ObservableCollection<Mode> _modeCollection = new ObservableCollection<Mode>();
private ObservableCollection<string> _registers = new ObservableCollection<string>();

private bool _isCondensed = false;
#endregion

#region Public Members
public string FileName => Path.GetFileName(this.FilePath);
public string FilePath
{
get { return _filePath; }
set { _filePath = value; OnPropertyChanged("FilePath"); OnPropertyChanged("FileName"); }
}



public string TimeSetUsed
{
get { return _timeSetItem; }
set { _timeSetItem = value; OnPropertyChanged("TimeSetUsed"); }
}
[XmlIgnore]
```

```
public DigiTimeObj TimeSetUsedObj
{
get { return _timeSetUsedObj; }
set
{
_timeSetUsedObj = value; OnPropertyChanged("TimeSetUsedObj");
if (value != null)
{
TimeSetUsed = value.Name;
foreach (Mode m in ModeCollection)
{
foreach (ModeVector mv in m.ModeVectors)
{
foreach (Operation op in mv.VectorizedData)
{
op.TimeSet = value.Name;
}
}
}
this.CompileAllModes();
}
}
}

public string ClockPinItem
{
get { return _clockPinItem; }
set { _clockPinItem = value; OnPropertyChanged("ClockPinItem"); }
}
[XmlIgnore]
public PinModel ClockPin
{
get { return _clockPin; }
set
{
_clockPin = value;  OnPropertyChanged("ClockPin");
if (value != null)
{
ClockPinItem = value.PinName;
this.CompileAllModes();
}
}
```

```
}

public string DataPinItem
{
get { return _dataPinItem; }
set { _dataPinItem = value; OnPropertyChanged("DataPinItem"); }
}
[XmlIgnore]
public PinModel DataPin
{
get { return _dataPin; }
set
{
_dataPin = value; OnPropertyChanged("DataPin");
if (value != null)
{
DataPinItem = value.PinName;
this.CompileAllModes();
}
}
}

//Columns
public ObservableCollection<string> Registers
{
get { return _registers; }
set { _registers = value; OnPropertyChanged("Registers"); }
}

public bool IsCondensed
{
get { return _isCondensed; }
set { _isCondensed = value; OnPropertyChanged("IsCondensed"); }
}

//Rows
public ObservableCollection<Mode> ModeCollection
{
get { return _modeCollection; }
set { _modeCollection = value; OnPropertyChanged("ModeCollection"); }
}
```

```
#endregion


#region Events
public event Action ChangeOccured;
public event Action CommitEditBeforeSave;
#endregion

#region Constructor
public DigitalPatternModel() { } //Deserialization Constructor
public DigitalPatternModel(string filePath)
{
this.FilePath = filePath;

//Temp.
ModeCollection.Add(new Mode() { Name = "Mode Name Here", RegData = new List<string>() });
}
#endregion

public void Save()
{
CommitEditBeforeSave?.Invoke();
this.SaveToXml(this.FilePath);
Console.WriteLine(string.Format("Saved: {0}", this.FilePath));
}
public void Load()
{

}
public bool Undo()
{
Console.WriteLine("Undo operation not implemented for Digital Timing Files");
return true;
}

public bool Redo()
{
Console.WriteLine("Undo operation not implemented for Digital Timing Files");
return true;
}

#region XML Import/Export
```

```
public void SaveToXml(string FileName)
{
try
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(FileName))
{
xmlSerializer.Serialize(writer, this);
}
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
}
public static DigitalPatternModel LoadFromXml(string FileName)
{
try
{
XmlSerializer ser = new XmlSerializer(typeof(DigitalPatternModel));
DigitalPatternModel digiPatternModelAfterDeSerialize;

using (StreamReader sr = new StreamReader(FileName))// Deserialize XML file.
{
digiPatternModelAfterDeSerialize = (DigitalPatternModel)ser.Deserialize(sr);
}

return digiPatternModelAfterDeSerialize;
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); return null; }
}
#endregion
public void RelinkPinData(PinMapModel projectPinMap)
{
foreach (Data.PinModel pin in projectPinMap.DigitalPins.Where(pin => pin.PinName ==
DataPinItem))
{
DataPin = pin;
}
foreach (Data.PinModel pin in projectPinMap.DigitalPins.Where(pin => pin.PinName ==
ClockPinItem))
{
ClockPin = pin;
```

```
}
}


public void GeneratePatternFiles(string FileName, string pinMapFilePath)
{
//Create a sub directory or make sure it exists to store all the generated pattern files.
string dir = Path.GetDirectoryName(FileName);
string folder = Path.GetFileNameWithoutExtension(FileName);
string OutputDir = Path.Combine(dir, folder);
if (!Directory.Exists(OutputDir))
{
Directory.CreateDirectory(OutputDir);
}

//Check if model compile requirements are empty.
if (string.IsNullOrEmpty(this.TimeSetUsed) || string.IsNullOrEmpty(this.ClockPinItem) ||
string.IsNullOrEmpty(this.DataPinItem))
{
Console.WriteLine($"{this.FileName}: Time Set, Clock Pin, and Data Pin cannot be empty .");
return;
}

//this.BuildAllDPatSrcDataForModes(); //Needed before save.
this.SaveDPATSRC(FileName, OutputDir);

//Check to see if the seDPinPatternCompiler_64.exe has been installed or exists in the installation
path.
string compilerDDSToolPath = @"C:\Program Files\IVI
Foundation\IVI\Drivers\seDPin\PatternCompiler\seDPinPatternCompiler_64.exe"; //Universal
Installation Path.
if (!File.Exists(compilerDDSToolPath)) //If exe file does not exist, throw a message.
{
MessageBox.Show($"{compilerDDSToolPath} does not exist or was never installed...");
return;
}
//Check to make sure the .pinmap file for SEDPIN usage exists and then use it as a parameter.
string SeDpinMapFilePath = Path.ChangeExtension(pinMapFilePath, ".pinmap");
if (!File.Exists(SeDpinMapFilePath))
{
MessageBox.Show($"{SeDpinMapFilePath} does not exist, cannot continue without a .pinmap
file");
```

```
return;
}
Compile_To_DPAT(FileName, SeDpinMapFilePath, OutputDir);
}
#region Compile to DPAT
// Define static variables shared by class methods.
private static StringBuilder ShellOutput = null;
private static StringBuilder ShellErrorsToWrite = null;
private static int numOutputLines = 0;
private void Compile_To_DPAT(string FileName, string SeDpinMapFilePath, string
OutputDirectory)
{
try
{
// create and start a Stopwatch instance
Stopwatch stopwatch = Stopwatch.StartNew();

#region Test Code
using (Process myProcess = new Process())
{
myProcess.StartInfo.FileName = @"powershell.exe";
myProcess.StartInfo.UseShellExecute = false; //Setting this to true will not allow us to redirect the
input and output.
myProcess.StartInfo.RedirectStandardInput = true;
myProcess.StartInfo.WorkingDirectory = @"C:\Program Files\IVI
Foundation\IVI\Drivers\seDPin\PatternCompiler"; //compilerDDSToolPath parent directory.
myProcess.StartInfo.RedirectStandardOutput = true;
myProcess.StartInfo.RedirectStandardError = true; //TEST
myProcess.StartInfo.CreateNoWindow = true;
// Set our event handler to asynchronously read the sort output.
myProcess.OutputDataReceived += OutputHandler;
//myProcess.ErrorDataReceived += ErrorHandler;

ShellOutput = new StringBuilder(); //Must be set on each call since var is static.
ShellErrorsToWrite = new StringBuilder(); //Must be set on each call since var is static.

myProcess.Start();
StreamWriter myStreamWriter = myProcess.StandardInput;

// Start the asynchronous read of the sort output stream.
myProcess.BeginOutputReadLine();
```

```
foreach (Mode mode in this.ModeCollection)
{
string AlteredModeName = Regex.Replace(mode.Name, "[^a-zA-Z0-9_]+", "_"); //Replace all
special characters in the mode name for compiler reasons "parenthesis".
string dPatSrcFilePath = Path.Combine(OutputDirectory, string.Format("{0}.dpatsrc",
AlteredModeName));

string outputFilePath = Path.Combine(OutputDirectory, string.Format("{0}.dpat",
AlteredModeName)); //.dpat file path

string Command = $@".\seDPinPatternCompiler_64.exe -p '{dPatSrcFilePath}' -m
'{SeDpinMapFilePath}' -o '{outputFilePath}' -d '{OutputDirectory}' -suppress_log";
myStreamWriter.WriteLine(Command);
}
myStreamWriter.Close();

// Wait for the sort process to write the sorted text lines.
myProcess.WaitForExit();
}
#endregion

//Write all shell output to the compile logs.
using (StreamWriter writetext = new StreamWriter(Path.Combine(OutputDirectory,
"AllCompileLogs.txt")))
{
writetext.Write(ShellOutput);
}
stopwatch.Stop();

#region Write final result to console
if (string.IsNullOrEmpty(ShellErrorsToWrite.ToString()))
{
Console.WriteLine($"Successfully compiled all .dpat files (in {stopwatch.ElapsedMilliseconds} ms)
using {FileName}");
}
else
{
Console.WriteLine(ShellErrorsToWrite.ToString());
Console.WriteLine($"Failed to compile all .dpat files using {FileName}");
}
#endregion Write final result to console
}
```

```
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
}
private static void OutputHandler(object sendingProcess,
System.Diagnostics.DataReceivedEventArgs outLine)
{
// Collect the sort command output.
if (!String.IsNullOrEmpty(outLine.Data))
{
numOutputLines++;

#region Error String Check
List<string> errorWords = new List<string>() { "[Failed]", "[failed]", "[Missing]", "[missing]", "[Error]",
"[error]" };
bool lineContainsError = errorWords.Any(c => outLine.Data.Contains(c));
if (lineContainsError)
{
//Console.WriteLine($"PowerShell: {outLine.Data}"); //Console.WriteLine() won't work since
standard input is redirected to the PowerShell process.
ShellErrorsToWrite.Append(Environment.NewLine + outLine.Data);
}
#endregion

// Add the text to the collected output.
ShellOutput.Append(Environment.NewLine + $"[{numOutputLines}] - {outLine.Data}");
}
}
#endregion

#region Translate to DPATSRC
public string CompileModeToDPATSRC(Mode mode)
{
StringBuilder sb = new StringBuilder();
try
{
// Build the pattern string
//sb.AppendLine("");
sb.AppendLine("dpat_format_version 1.1;");
sb.AppendLine(string.Format("timeset {0};", this.TimeSetUsed));
sb.AppendLine("svm_only;");
```

```
sb.AppendLine("");
sb.AppendLine("pattern " + mode.Name + " (" + this.ClockPinItem + ":b," + this.DataPinItem +
":b)");
sb.AppendLine("{");
//writetext.WriteLine("start:"); //Causes problem on update for loading sub-patterns or something
on the instrument.

//Append sections of pattern.
foreach (ModeVector vector in mode.ModeVectors)
{
int modeVectorIndex = mode.ModeVectors.IndexOf(vector);
ObservableCollection<Operation> MasterOps = vector.VectorizedData;

//NoNeed or Empty bypass
if (MasterOps.Count() <= 0)
{
continue;
}

//Sort sections of the operations w/ LINQ.
List<Operation> startSequence = MasterOps.Where(x => x.Section ==
VectorSection.Start_Sequence_Control).ToList();
List<Operation> commandSequence = MasterOps.Where(x => x.Section ==
VectorSection.Command).ToList();
List<Operation> dataSequence = MasterOps.Where(x => x.Section ==
VectorSection.Data).ToList();
List<Operation> busPark = MasterOps.Where(x => x.Section ==
VectorSection.Bus_Park).ToList();

//Being Writing the DPATSRC.
sb.AppendLine("// Write reg " + this.Registers[modeVectorIndex] + " data " +
mode.ModeVectors[modeVectorIndex].InputValue);
//Print condensed.
if (this.IsCondensed)
{
sb.AppendLine("// Start sequence control");
AppendOperationSequence(sb, startSequence);
sb.AppendLine("// Command");
AppendOperationSequence(sb, commandSequence);
sb.AppendLine("// Data");
AppendOperationSequence(sb, dataSequence);
sb.AppendLine("// Bus Park");
```

```
AppendOperationSequence(sb, busPark);
}
else //Print expanded.
{
sb.AppendLine("// Start sequence control");
foreach (var sVector in startSequence)
{
sb.AppendLine(sVector.ToString());
}
sb.AppendLine("// Command");
foreach (var cVector in commandSequence)
{
sb.AppendLine(cVector.ToString());
}
sb.AppendLine("// Data");
foreach (var dVector in dataSequence)
{
sb.AppendLine(dVector.ToString());
}
sb.AppendLine("// Bus Park");
foreach (var bpVector in busPark)
{
sb.AppendLine(bpVector.ToString());
}
}

}

sb.AppendLine($"HALT\t\t\t\t\t\t  >\t{"-"}\t\tX{"  "}X;"); //Extra HALT line to fix "timing generator"
error (forwarded email from jerry).
sb.AppendLine("}");

return sb.ToString();
}
catch (Exception ex)
{
Console.WriteLine("Vectors failed to compile...");
return sb.ToString();
}
}
public void Compile_Single_Mode_(Mode modeToBuild)
{
```

```
//Compiles mode's Vectorized data into DPATSRC
modeToBuild.DPatSrcData = CompileModeToDPATSRC(modeToBuild);
}
public void CompileAllModes()
{
foreach (Mode mode in this.ModeCollection)
{
//Compiles all Vectorized data into DPATSRC for every mode.
mode.DPatSrcData = CompileModeToDPATSRC(mode);
}
}

//Takes the input values from data vectors and builds/sets the operations needed to compile the
DPATSRC vectors into a pattern text.
#region Vectorize Overload Method Group
public void Vectorize_All_Modes_For_Pattern()
{
foreach (Mode mode in this.ModeCollection)
{
Vectorize_All_For_Mode(mode);
}
}
public void Vectorize_All_For_Mode(Mode mode)
{
foreach (ModeVector vector in mode.ModeVectors)
{
this.Vectorize_Single(mode, vector);
}
}

public void Vectorize_Single(ModeVector modeVector, string register)
{

ObservableCollection<Operation> MasterOps = modeVector.VectorizeInput(register,
this.TimeSetUsed);
modeVector.VectorizedData.Clear();
foreach (Operation op in MasterOps)
{
modeVector.VectorizedData.Add(op);
}
}
public void Vectorize_Single(ModeVector modeVector, int registerIndex)
```

```
{
ObservableCollection<Operation> MasterOps =
modeVector.VectorizeInput(this.Registers[registerIndex], this.TimeSetUsed);
modeVector.VectorizedData.Clear();
foreach (Operation op in MasterOps)
{
modeVector.VectorizedData.Add(op);
}
}
public void Vectorize_Single(Mode parentMode, ModeVector modeVector)
{
int modeVectorIndex = parentMode.ModeVectors.IndexOf(modeVector);
ObservableCollection<Operation> MasterOps =
modeVector.VectorizeInput(this.Registers[modeVectorIndex], this.TimeSetUsed);
modeVector.VectorizedData.Clear();
foreach (Operation op in MasterOps)
{
modeVector.VectorizedData.Add(op);
}
}
# endregion Vectorize Overload Method Group

//Saves the DPATSRC text data from every mode to it's own pattern file.
private void SaveDPATSRC(string FileName, string OutputDirectory)
{
try
{
foreach (Mode mode in this.ModeCollection)
{
string NewModeName = Regex.Replace(mode.Name, "[^a-zA-Z0-9_]+", "_"); //Replace all special
characters in the mode name for compiler reasons "parenthesis".
string NewFilePath = Path.Combine(OutputDirectory, string.Format("{0}.dpatsrc",
NewModeName));
using (StreamWriter writetext = new StreamWriter(NewFilePath))
{
mode.DPatSrcData = CompileModeToDPATSRC(mode);
writetext.Write(mode.DPatSrcData);
}
}
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
```

```
}

#region Old non dynamic DPATSRC Translation code
///private void SaveDPATSRC(string FileName, string OutputDirectory)
///{
///    try
///    {
///        // create and start a Stopwatch instance
///        Stopwatch stopwatch = Stopwatch.StartNew();
///
///        foreach (Mode mode in this.ModeCollection)
///        {
///            string NewModeName = Regex.Replace(mode.Name, "[^a-zA-Z0-9_]+", "_"); //Replace all special characters in the mode name for compiler reasons "parenthesis".
///            string NewFilePath = Path.Combine(OutputDirectory, string.Format("{0}.dpatsrc", NewModeName));
///            using (StreamWriter writetext = new StreamWriter(NewFilePath))
///            {
///                writetext.WriteLine("");
///                writetext.WriteLine("dpat_format_version 1.1;");
///                writetext.WriteLine(string.Format("timeset {0};", this.TimeSetUsed));
///                writetext.WriteLine("svm_only;");
///                writetext.WriteLine("");
///                writetext.WriteLine(string.Format("pattern {0} ({1}:b,{2}:b)", NewModeName, this.ClockPinItem, this.DataPinItem));
///                writetext.WriteLine("{");
///                //writetext.WriteLine("start:"); //Causes problem on update for loading sub-patterns or something on the instrument.
///                for (int dataIndex = 0; dataIndex < mode.RegData.Count; dataIndex++)
//foreach(string data in mode.RegData)
///                {
///                    string data = mode.RegData[dataIndex];
///                    string register = Regex.Replace(Registers[dataIndex], "_", "", RegexOptions.Compiled); //[^a - zA - Z0 - 9_.]+
///
///                    #region Pre-Process Data Checks
///    Break if the data is not required. Should break if the data string does not contain an x or perhaps a 0x : TBD.
///    if (data == "no need" || data == "No Need" || data == "NN" || data == "nn") { continue; }
///                    else if (string.IsNullOrEmpty(data)) //Break if the data is null or empty then send a warning to the user.
///                    {
```

```
///                 Console.WriteLine($"WARNING: {Path.GetFileName(FileName)}: Mode
{mode.Name} contains a blank hex value for Register {register}"); continue;
///             }
///             //For some reason it allows the letter "a" through and breaks the entire process.
///             if (!OnlyHexInString(data)) //Checks if the data only contains only C-style hex
notation (0xFF) values
///             {
///                 Console.WriteLine($"WARNING: {Path.GetFileName(FileName)}: Mode
{mode.Name} contains invalid hex data for Register {register}"); continue;
///             }
///             #endregion Pre-Process Data Checks
///
///             writetext.WriteLine(string.Format("// Write reg {0} data {1}", register, data));
///             writetext.WriteLine("// Start sequence control");
///             writetext.WriteLine($"REPEAT(3)                 >  {this.TimeSetUsed}     0   X;");
///             writetext.WriteLine($"REPEAT(6)                 >  {this.TimeSetUsed}     0   0;");
///             writetext.WriteLine($"                          >  {this.TimeSetUsed}     0   1;");
///             writetext.WriteLine($"                          >  {this.TimeSetUsed}     0   0;");
///             writetext.WriteLine("//Command");
///             writetext.WriteLine($"REPEAT(4)                 >  {this.TimeSetUsed}     1   1;");
//Slave address
///             int[] cArray = TranslateToBinary(register, "reg");
///             for (int i = 0; i < cArray.Length; i++) //reg address C7, C6, C5, C4, C3, C2, C1, C0
///             {
///                 writetext.WriteLine($"                      >  {this.TimeSetUsed}     1
{cArray[i]};");
///             }
///             int cParityBit = CalculateParityBit(cArray);
///             writetext.WriteLine($"                          >  {this.TimeSetUsed}     1
{cParityBit};");
///             writetext.WriteLine("//Data");
///             int[] dArray = TranslateToBinary(data, "data");
///             for (int i = 0; i < dArray.Length; i++)//data D7, D6, D5, D4, D3, D2, D1, D0
///             {
///                 writetext.WriteLine($"                      >   {this.TimeSetUsed}      1
{dArray[i]};");
///             }
///             int dParityBit = CalculateParityBit(dArray);
///             writetext.WriteLine($"                          >  {this.TimeSetUsed}     1
{dParityBit};"); //Write in the parity bit.
///             writetext.WriteLine("//Bus Park");
///             writetext.WriteLine($"                          >  {this.TimeSetUsed}     1   0;");
```

```
///                 writetext.WriteLine($"                          >  {this.TimeSetUsed}     0   X;");
///                 writetext.WriteLine($"                          >  {this.TimeSetUsed}     X   X;");
///             }
///             writetext.WriteLine($"HALT                          >  {"-"}        X   X;"); //Extra HALT line
to fix "timing generator" error (forwarded email from jerry).
///             writetext.WriteLine("}");
///         }
///     }
///     stopwatch.Stop();
///     Console.WriteLine($"Successfully compiled all modes to .dpatsrc files (in
{stopwatch.ElapsedMilliseconds} ms) using {this.FilePath}");
///  }
///  catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
///}

////Counts the total number of ones in the 8 bit array given to determine and return the parity bit.
//public static int CalculateParityBit(int[] bitArray)
//{
//  int totalOnes = 0;
//  for (int i = 0; i < bitArray.Length; i++)
//  {
//      totalOnes += bitArray[i] == 1 ? 1 : 0;
//  }
//  int resultVal = totalOnes % 2 == 0 ? 1 : 0;  //odd number returns 0 as parity bit while even
numbers return 1 as parity bit
//  return resultVal;
//}
//
////Translates the register codes and the data codes to return the data in the order it should be
serialized.
//public static int[] TranslateToBinary(string asciiCode, string regORdata)
//{
//  int[] returnValue;
//
//  char[] code = asciiCode.ToCharArray();
//  switch (regORdata)
//  {
//      case "reg":
//          returnValue = new int[8];
//          returnValue[0] = 0; //command C7  (010: Write, 011: Read)
//          returnValue[1] = 1; //command C6
//          returnValue[2] = 0; //command C5
```

```
//            char[] cBinChars = hex2binary((code[2].ToString() + code[3].ToString())).ToCharArray();
//            for (int i = 3; i < cBinChars.Length; i++)
//            {
//                returnValue[i] = Int32.Parse(cBinChars[i].ToString()) ;
//            }
//            break;
//        case "data":
//            returnValue = new int[8];
//            char[] dBinChars = hex2binary((code[2].ToString() + code[3].ToString())).ToCharArray();
//            for (int i = 0; i < dBinChars.Length; i++)
//            {
//                returnValue[i] = Int32.Parse(dBinChars[i].ToString());
//            }
//            break;
//        default: returnValue = null; break;
//    }
//
//    return returnValue;
//}
////Helper method converting hex to binary.
//public static string hex2binary(string hexvalue)
//{
//    String result = hexvalue
//    .Aggregate(new StringBuilder(), (builder, c) =>
//    builder.Append(Convert.ToString(Convert.ToInt32(c.ToString(), 16), 2).PadLeft(4, '0')))
//    .ToString();
//
//    return result;
//}

//Helper method.
#endregion Old non dynamic DPATSRC Translation code
public static bool OnlyHexInString(string data)
{
// For C-style hex notation (0xFF) you can use @"\A\b(0[xX])?[0-9a-fA-F]+\b\Z"
return Regex.IsMatch(data, @"\A\b(0[xX])?[0-9a-fA-F]+\b\Z");
}

// Appends a sequence of operations to the StringBuilder, using REPEAT opcodes where possible
public void AppendOperationSequence(StringBuilder sb, List<Operation> sequence)
{
try
```

```
{
if (sequence != null && sequence.Count() > 0)
{
int repeatCount = 1;
for (int i = 1; i < sequence.Count; i++)
{
if (sequence[i].RepeatEquals(sequence[i - 1]))
{
repeatCount++;
}
else
{
if (repeatCount > 1)
{
sequence[i - 1].Opcode = "REPEAT(" + repeatCount + ")";
sb.AppendLine(sequence[i - 1].ToString());
sequence[i - 1].Opcode = "";
}
else
{
sb.AppendLine(sequence[i - 1].ToString());
}
repeatCount = 1;
}
}
if (repeatCount > 1)
{
sequence[sequence.Count - 1].Opcode = "REPEAT(" + repeatCount + ")";
sb.AppendLine(sequence[sequence.Count - 1].ToString());
sequence[sequence.Count - 1].Opcode = "";
}
else
{
sb.AppendLine(sequence[sequence.Count - 1].ToString());
}
}
}
catch (Exception ex)
{
Console.WriteLine($"Repeats failed to condense: {ex.Message}");
}
}
```

```
#endregion

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
OnChangeOccured();
}
#endregion

public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}

}

}
```

DigitalTimingModel.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Xml;
using System.Xml.Linq;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels
{

[Serializable]
public class DigiTimeObj : INotifyPropertyChanged
{
```

```
public string _timeSetName;
public string _pinItem;
private PinModel _digiPin = null;
private double _period = 0;

public string Name //Creates "TimeSet"
{
get { return _timeSetName; }
set { _timeSetName = value; OnPropertyChanged("Name"); }
}
public double Period  // In nanoseconds (ns).
{
get { return _period; }
set { _period = value; OnPropertyChanged("Period"); }
}

[XmlIgnore]
public PinModel DigiPin
{
get { return _digiPin; }
set
{
_digiPin = value; OnPropertyChanged("DigiPin");
if(value != null)
{
PinItem = value.PinName;
}
}
}
public string PinItem //PinName
{
get { return _pinItem; }
set { _pinItem = value; OnPropertyChanged("PinItem"); }
}

public string EdgeMultiplier { get; set; } //1x or 2x.
public string DriveFormat { get; set; } //NR, RL, RH, SBC.
public int DriveOn { get; set; } ///Assuming 0 = OFF & 1 = ON.
public double DriveData { get; set; }
public double DriveReturn { get; set; } // In nanoseconds (ns).
public double DriveOff { get; set; } // In nanoseconds (ns).
public double CompareStrobe { get; set; } // In nanoseconds (ns).
```

```
public string Comment { get; set; }

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
#endregion
}

[Serializable] //[XmlRoot("TimingFile")]
public class DigitalTimingModel : INotifyPropertyChanged, IFileData
{
#region Private members
private string _fileName;
private string _filePath;
private ObservableCollection<DigiTimeObj> _source = new
ObservableCollection<DigiTimeObj>();
#endregion

#region Public members
public string FileName => Path.GetFileName(this.FilePath);
public string FilePath
{
get { return _filePath; }
set { _filePath = value; OnPropertyChanged("FilePath"); OnPropertyChanged("FileName"); }
}
public ObservableCollection<DigiTimeObj> DigiTimeSource
{
get { return _source; }
set { _source = value; OnPropertyChanged("Source"); }
}
#endregion

#region Events
public event Action ChangeOccured;
public event Action CommitEditBeforeSave;
#endregion

#region Constructor
```

```
public DigitalTimingModel() { } //Deserialization Constructor
public DigitalTimingModel(string filePath)
{
this.FilePath = filePath;
this.DigiTimeSource.Add(new DigiTimeObj());
}
#endregion

#region Methods
public List<DigiTimeObj> GetTimeSets()
{
List<string> nameContainsCheckList = new List<string>(); //Temp List for name checks.
List<DigiTimeObj> rVal = new List<DigiTimeObj>();
foreach (var timeobj in this.DigiTimeSource)
{
if (!nameContainsCheckList.Contains(timeobj.Name)) //if list does not contain time set name, then
add it.
{
nameContainsCheckList.Add(timeobj.Name);
rVal.Add(timeobj);
}
}
return rVal;
}
#endregion

public void Save()
{
CommitEditBeforeSave?.Invoke();
this.Save_NI_XML(this.FilePath);
Console.WriteLine(string.Format("Saved: {0}", this.FilePath));
}
public void Load()
{

}

public bool Undo()
{
Console.WriteLine("Undo operation not implemented for Digital Timing Files");
return true;
}
```

```
public bool Redo()
{
Console.WriteLine("Redo operation not implemented for Digital Timing Files");
return true;
}

#region XML Import/Export
public void Save_NI_XML(string FileName)
{
try
{
List<string> timeSets = DigiTimeSource.Select(x => x.Name).Distinct().ToList();

//string NiFilePath = Path.Combine( Path.GetDirectoryName(FileName),
$"{Path.GetFileNameWithoutExtension(FileName)} (Compile){Path.GetExtension(FileName)}");
using (XmlTextWriter xmlWriter = new XmlTextWriter(FileName, System.Text.Encoding.UTF8) {
Formatting = Formatting.Indented })
{
xmlWriter.WriteStartDocument();
xmlWriter.WriteStartElement("TimingFile");
xmlWriter.WriteAttributeString("xmlns:xsd", "http://www.w3.org/2001/XMLSchema");
xmlWriter.WriteAttributeString("xmlns:xsi", "http://www.w3.org/2001/XMLSchema-instance");
xmlWriter.WriteAttributeString("schemaVersion", "1.0");
xmlWriter.WriteAttributeString("xmlns", "http://www.ni.com/Semiconductor/Timing");

xmlWriter.WriteStartElement("TimingSheet");
xmlWriter.WriteStartElement("TimeSets");
foreach (string set in timeSets)
{
if (!string.IsNullOrEmpty(set)) //if string is not null or empty ("") then continue.
{
xmlWriter.WriteStartElement("TimeSet");
xmlWriter.WriteAttributeString("name", set);
List<DigiTimeObj> times = DigiTimeSource.Where(x => x.Name == set).ToList();  ///Get list of
DigiTimeObj's with the same "time set" name.
xmlWriter.WriteElementString("Period", $"{times.FirstOrDefault().Period} ns");
xmlWriter.WriteStartElement("PinEdges");
foreach (DigiTimeObj dto in times) //Missing Edge multiplier thing.
{
if (dto.DigiPin != null) //If the pin item of the pin egde is not null
{
```

```
xmlWriter.WriteStartElement("PinEdge");
xmlWriter.WriteAttributeString("pin", dto.PinItem);
switch (dto.DriveFormat)
{
case "NR": //Only differences is drive format element name and NR has no return element.
xmlWriter.WriteStartElement("DriveNonReturn");
xmlWriter.WriteElementString("On", dto.DriveOn.ToString());
xmlWriter.WriteElementString("Data", $"{dto.DriveData} ns");
xmlWriter.WriteElementString("Off", $"{dto.DriveOff} ns");
xmlWriter.WriteEndElement();
break;
case "RL":
xmlWriter.WriteStartElement("ReturnToLow");
xmlWriter.WriteElementString("On", dto.DriveOn.ToString());
xmlWriter.WriteElementString("Data", $"{dto.DriveData} ns");
xmlWriter.WriteElementString("Return", $"{dto.DriveReturn} ns");
xmlWriter.WriteElementString("Off", $"{dto.DriveOff} ns");
xmlWriter.WriteEndElement();
break;
case "RH":
xmlWriter.WriteStartElement("ReturnToHigh");
xmlWriter.WriteElementString("On", dto.DriveOn.ToString());
xmlWriter.WriteElementString("Data", $"{dto.DriveData} ns");
xmlWriter.WriteElementString("Return", $"{dto.DriveReturn} ns");
xmlWriter.WriteElementString("Off", $"{dto.DriveOff} ns");
xmlWriter.WriteEndElement();
break;
case "SBC":
xmlWriter.WriteStartElement("SurroundByComplement");
xmlWriter.WriteElementString("On", dto.DriveOn.ToString());
xmlWriter.WriteElementString("Data", $"{dto.DriveData} ns");
xmlWriter.WriteElementString("Return", $"{dto.DriveReturn} ns");
xmlWriter.WriteElementString("Off", $"{dto.DriveOff} ns");
xmlWriter.WriteEndElement();
break;
}
xmlWriter.WriteStartElement("CompareStrobe");
xmlWriter.WriteElementString("Strobe", $"{dto.CompareStrobe} ns");
xmlWriter.WriteEndElement();

xmlWriter.WriteElementString("DataSource", "Pattern"); //???
xmlWriter.WriteEndElement();
```

```
}
else { Console.WriteLine($"{FileName}: Time Set {set} cannot have serialize a Pin Edge using a null Pin Item."); }
}
xmlWriter.WriteEndElement();
xmlWriter.WriteEndElement();
}
}
xmlWriter.WriteEndElement();
xmlWriter.WriteEndElement();

xmlWriter.WriteEndElement();
xmlWriter.WriteEndDocument();
}
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
}
public static DigitalTimingModel Load_NI_Xml(string FileName)
{
DigitalTimingModel TimingModel = new DigitalTimingModel() { FilePath = FileName };

XDocument document = XDocument.Load(FileName);
foreach (XElement TimingSheets in document.Root.Elements()) //TimingFile(root) => TimingSheet(s)
{
foreach (XElement TimeSets in TimingSheets.Elements()) //TimeSets.
{
foreach(XElement TimeSet in TimeSets.Elements()) //TimeSet (Each)
{
//Create each TimeSet / DigiTimeObj for the model collection.
string timeSetName = TimeSet.Attributes().FirstOrDefault().Value;
double period = 0;
List<DigiTimeObj> timeSetPinEdgeObjs = new List<DigiTimeObj>(); //Allows the creation of multiple objects per "TimeSet".

foreach(XElement timeSetElement in TimeSet.Elements())
{
switch (timeSetElement.Name.LocalName)
```

```
{
case "Period":
string periodString = TimeSet.Element(timeSetElement.Name).Value;
string[] periodStringSplit = periodString.Split(null); //Splits by whitespace by default.
period = double.Parse(periodStringSplit[0]); //Splits the "ns" out of the data. (TEMP FIX).
break;
case "PinEdges":
foreach (XElement pinEdge in timeSetElement.Elements()) //PinEdge (each).
{
string pinName = pinEdge.Attributes().FirstOrDefault().Value;
var dto = new DigiTimeObj() { Name = timeSetName, Period = period, PinItem = pinName,
EdgeMultiplier = "1x" };

foreach (XElement PECompnent in pinEdge.Elements())
{
switch (PECompnent.Name.LocalName)
{
case "DriveNonReturn":
case "ReturnToLow":
case "ReturnToHigh":
case "SurroundByComplement":
dto.DriveFormat =
DigitalTimingModel.DriveFormatStringConverter(PECompnent.Name.LocalName);
foreach (XElement PEComponentElement in PECompnent.Elements())
{
switch (PEComponentElement.Name.LocalName)
{
case "On":
dto.DriveOn = int.Parse(PECompnent.Element(PEComponentElement.Name).Value);
break;
case "Data":
dto.DriveData =
double.Parse(PECompnent.Element(PEComponentElement.Name).Value.Split(null).FirstOrDefaul
t());
break;
case "Return": //DriveNonReturn case: should not hit this.
dto.DriveReturn =
double.Parse(PECompnent.Element(PEComponentElement.Name).Value.Split(null).FirstOrDefaul
t());
break;
case "Off":
dto.DriveOff =
```

```
double.Parse(PECompnent.Element(PEComponentElement.Name).Value.Split(null).FirstOrDefaul
t());
break;
}
}
break;

case "CompareStrobe":
foreach (XElement PEComponentElement in PECompnent.Elements())
{
if(PEComponentElement.Name.LocalName == "Strobe")
{
dto.CompareStrobe =
double.Parse(PECompnent.Element(PEComponentElement.Name).Value.Split(null).FirstOrDefaul
t());
}
}
break;
case "Comment":
foreach (XElement PEComponentElement in PECompnent.Elements())
{
if (PEComponentElement.Name.LocalName == "Comment")
{
dto.Comment = PECompnent.Element(PECompnent.Name).Value;
}
}
break;
default:
//Do Nothing, Like for <DataSource>Pattern</DataSource>.
break;
}
}

timeSetPinEdgeObjs.Add(dto);
}
break;
}
}

//Put all the TimeSet-PinEgde-Objects in the TimingModel source collection.
foreach (DigiTimeObj dtoToAdd in timeSetPinEdgeObjs)
{
```

```
TimingModel.DigiTimeSource.Add(dtoToAdd);
}
}
}
}

return TimingModel;
}

public static string DriveFormatStringConverter(string stringToConvert)
{
switch (stringToConvert)
{
case "DriveNonReturn": return "NR";
case "ReturnToLow": return "RL";
case "ReturnToHigh": return "RH";
case "SurroundByComplement": return "SBC";
case "NR": return "DriveNonReturn";
case "RL": return "ReturnToLow";
case "RH": return "ReturnToHigh";
case "SBC": return "SurroundByComplement";
default: Console.WriteLine("DriveFormatStringConverter: StringConversionFailed"); return
string.Empty;
}
}
#endregion

public void RelinkPinData(PinMapModel projectPinMap)
{
foreach (var digiTime in this.DigiTimeSource)
{
foreach (Data.PinModel pin in projectPinMap.DigitalPins.Where(pin => pin.PinName ==
digiTime.PinItem))
{
digiTime.DigiPin = pin;
}
}
}

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
```

```
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
OnChangeOccured();
}
#endregion

public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
}
}
```

GenericRffePatternModel.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels
{
//public class OpcodeTag
//{
//
//}

[Serializable]
public class GenericMode : INotifyPropertyChanged
{
#region Private Members
private string _name = string.Empty;
```

```
private List<string> _registers = new List<string>();
private string _dPatSrcData = string.Empty;
private ObservableCollection<GenericModeVector> _modeVectors = new
ObservableCollection<GenericModeVector>();
#endregion

#region Public Members
public string Name
{
get { return _name; }
set { _name = value; OnPropertyChanged("Name"); }
}

//OLD here for backwards compatibility.
public List<string> RegData //Data For Register. //REPLACED BY ModeVectors
{
get { return _registers; }
set { _registers = value; OnPropertyChanged("RegData"); }
}

public ObservableCollection<GenericModeVector> ModeVectors
{
get { return _modeVectors; }
set { _modeVectors = value; OnPropertyChanged("ModeVectors"); }
}

private ValidationStatus _Status = ValidationStatus.Valid;
public ValidationStatus Status ///Check the name to ensure there are no names with paranthesis in
it.
{
get { return _Status; }
set { _Status = value; OnPropertyChanged("Status"); }
}
private string _errorMessage = string.Empty;
public string ErrorMessage
{
get { return _errorMessage; }
set { _errorMessage = value; OnPropertyChanged("ErrorMessage"); }
}

[XmlIgnore]
public bool IsExpandable => ModeVectors.Count() > 0 ? true : false;
```

```
public bool IsExpanded { get; set; }

private ObservableCollection<Operation> _opcodesDetected = new
ObservableCollection<Operation>();
[XmlIgnore]
public ObservableCollection<Operation> OpcodesDetected
{
get { return _opcodesDetected; }
set { _opcodesDetected = value; OnPropertyChanged("OpcodesDetected"); }
}

[XmlIgnore]
public string DPatSrcData
{
get { return _dPatSrcData; }
set { _dPatSrcData = value; OnPropertyChanged("DPatSrcData"); }
}

#endregion

#region Constructor
public GenericMode()
{

}
#endregion

public void FindDetectedOpcodes()
{
this.OpcodesDetected.Clear();

foreach (GenericModeVector mv in this.ModeVectors)
{
foreach (Operation op in mv.VectorizedData)
{
if (!string.IsNullOrEmpty(op.Opcode))
{
OpcodesDetected.Add(op);
}
}
}
```

```
OnPropertyChanged("IsExpandable");
}

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
#endregion
}

[Serializable]
public class GenericModeVector : INotifyPropertyChanged
{
private string _userIDValue = "F";
private string _registerValue = "00";
private string _maskValue = "FF";
private string _inputValue = "00";
private ObservableCollection<Operation> _vectorizedData = new
ObservableCollection<Operation>();


private RffePatternSpec _spec = RffePatternSpec.W; //Defualt Write (W)
public RffePatternSpec Spec
{
get { return _spec; }
set { _spec = value; OnPropertyChanged("Spec"); this.CheckPointStatus(); }
}

//Slave Address / User ID
public string UserIDValue
{
get { return _userIDValue; }
set { _userIDValue = value; OnPropertyChanged("UserIDValue"); this.CheckPointStatus(); }
}

public string _byteCountBinary = "0000";
public string ByteCountBinary
{
get { return _byteCountBinary; }
```

```
set { _byteCountBinary = value; OnPropertyChanged("ByteCountBinary");
this.CheckPointStatus(); }
}

public string RegisterValue
{
get { return _registerValue; }
set { _registerValue = value; OnPropertyChanged("RegisterValue"); this.CheckPointStatus(); }
}

//Mask Write Mask value (8-bit)
public string MaskValue
{
get { return _maskValue; } set { _maskValue = value; OnPropertyChanged("MaskValue");
this.CheckPointStatus(); }
}

//Data
public string InputValue
{
get { return _inputValue; }
set
{
_inputValue = value;
OnPropertyChanged("InputValue");
this.CheckPointStatus();
}
}

private ValidationStatus _Status = ValidationStatus.Valid;
public ValidationStatus Status ///Check bit/byte counts against the PatternSpec specifications to
ensure it's correct.
{
get { return _Status; }
set { _Status = value; OnPropertyChanged("Status"); }
}
private string _errorMessage = string.Empty;
public string ErrorMessage
{
get { return _errorMessage; }
set { _errorMessage = value; OnPropertyChanged("ErrorMessage"); }
}
```

```
//Calculated on import and edits.
public ObservableCollection<Operation> VectorizedData
{
get { return _vectorizedData; }
set { _vectorizedData = value; OnPropertyChanged("VectorizedData"); }
}

#region Constructors
public GenericModeVector() { } //Deserialization Constructor.
public GenericModeVector(string inputValue)
{
this.InputValue = inputValue;
}
#endregion

#region Methods

/// <summary>
/// Vectorizes the input into a collection of operations that are written out into a DPATSRC file in the correctly sectioned format.
/// </summary>
/// <param name="timeSet"></param>
/// <returns></returns>
public ObservableCollection<Operation> VectorizeInput(string timeSet)
{
List<Operation> MasterOps = new List<Operation>();

if(this.Status != ValidationStatus.Invalid)
{
try
{
string data = this.InputValue;
string register = Regex.Replace(this.RegisterValue, "_", "", RegexOptions.Compiled); //[^a - zA - Z0 - 9_.]+ //OLD

//unused.
#region Data Checks
/////Break if the data is not required. Should break if the data string does not contain an x or perhaps a 0x : TBD.
///if (data == "no need" || data == "No Need" || data == "NN" || data == "nn" || string.IsNullOrEmpty(data))
```

```
///{
///    return new ObservableCollection<Operation>(MasterOps);
///}

/////BUG: For some reason it allows the letter "a" through and breaks the entire process. (Caps
letters only ??)
/////Checks if the data only contains only C-style hex notation (0xFF) values
///if (!DigitalPatternModel.OnlyHexInString(data))
///{
///    Console.WriteLine($"RFFE input invalid: {data}, invalid hex data for Register {register}");
///    return new ObservableCollection<Operation>(MasterOps);
///}
#endregion Data Checks

///Uses string interpolation to determine which pattern spec to use.
//KeyValuePair<RffePatternSpec, string> RffePtrnSpecKvP =
ContainsAnyRffePatternSpec(this.InputValue);
//data = data.Replace(RffePtrnSpecKvP.Value, ""); ///Identifier removed (should contain Slave
Address, Register, and Data //Command is implied by identifier KvP above.
register = register.Replace("0x", ""); //Just incase.
data = data.Replace("0x", ""); //Just incase.

// Create a static sequence of operations for the start sequence
List<Operation> startSequence = new List<Operation>()
{
new Operation(timeSet, 0.ToString(), "X") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data }, //REPEAT(3)
new Operation(timeSet, 0.ToString(), "X") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "X") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },

new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data }, //REPEAT(6)
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
```

```
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },

new Operation(timeSet, 0.ToString(), "1") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "0") { Section = VectorSection.Start_Sequence_Control,
Comment = "//SSC", Original_Input = data }
};

///DYNAMIC
List<Operation> command = new List<Operation>();
List<Operation> registerSequence = new List<Operation>();
List<Operation> maskEightBit = new List<Operation>();
List<Operation> dataSequence = new List<Operation>();
switch (this.Spec)
{
case RffePatternSpec.W:
#region Write

// Convert the register and data values to binary strings according to pattern spec / bit
requirements.
string userIdBinary = HexToBinary(UserIDValue);
string registerBinary = HexToBinary(register);
string dataBinary = HexToBinary(data);

//Dynamic slave address
for (int i = 0; i < 4; i++) //reg address: SA3, SA2, SA1, SA0
{
command.Add(new Operation(timeSet, 1.ToString(), userIdBinary[i].ToString()) { Comment =
"//SA" + (userIdBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input = data
});
}

// Create a static, spec specific, sequence of operations for the command.
string writeCommandValues = "010 "; //As seen below. (Non-dynamic initial sequence)
//Read (0 1 1) & [ Write (0 1 0) ]
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C7", Section =
VectorSection.Command, Original_Input = data }); //C7 //Sets to Write //(0 1 0: Write --- 0 1 1:
Read)
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C6", Section =
VectorSection.Command, Original_Input = data }); //C6 //Sets to Write
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C5", Section =
```

```
VectorSection.Command, Original_Input = data }); //C5 //Sets to Write

for (int i = 3; i < 8; i++) //reg address: C4, C3, C2, C1, C0
{
command.Add(new Operation(timeSet, 1.ToString(), registerBinary[i].ToString()) { Comment =
"//C" + (registerBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input = data
});
}
//Append slave address bits and write/read bits (first 3 C7,6,5 bits) then Cut off the first 3 bits of
registerBinary before calculating the parity bit
string commadFrameBinary = userIdBinary + writeCommandValues + registerBinary.Substring(3);
//registerBinary.Substring(3) gets rid of the first 3 bit/chars to allow space for the command values
in the 8 bit commad pay load.
int commandParityBit = commadFrameBinary.Count(c => c == '1') % 2; //command
string commandParityBitString = commandParityBit == 0 ? "1" : "0"; //command
command.Add(new Operation(timeSet, 1.ToString(), commandParityBitString) { Comment = "//P",
Section = VectorSection.Command, Original_Input = data }); //Add in the parity bit.

// Create a sequence of operations for the data
for (int i = 0; i < 8; i++) //data: D7, D6, D5, D4, D3, D2, D1, D0
{
dataSequence.Add(new Operation(timeSet, 1.ToString(), dataBinary[i].ToString()) { Comment =
"//D" + (dataBinary.Length - (i + 1)), Section = VectorSection.Data, Original_Input = data });
}
// Calculate the parity bit for the data and command
int dataParityBit = dataBinary.Count(c => c == '1') % 2; //data
string dataParityBitString = dataParityBit == 0 ? "1" : "0"; //data
dataSequence.Add(new Operation(timeSet, 1.ToString(), dataParityBitString) { Comment = "//P",
Section = VectorSection.Data, Original_Input = data }); //Write in the parity bit.
#endregion
break;
case RffePatternSpec.R:
#region Read

// Convert the register and data values to binary strings according to pattern spec / bit
requirements.
string rUserIdBinary = HexToBinary(UserIDValue);
string rRegisterBinary = HexToBinary(register);
string rDataBinary = HexToBinary(data);

//Dynamic slave address
for (int i = 0; i < 4; i++) //reg address: SA3, SA2, SA1, SA0
```

```
{
command.Add(new Operation(timeSet, 1.ToString(), rUserIdBinary[i].ToString()) { Comment =
"//SA" + (rUserIdBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input = data
});
}

// Create a static, spec specific, sequence of operations for the command.
string readCommandValues = "011 "; //As seen below. (Non-dynamic initial sequence)
//Read (0 1 1)
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C7", Section =
VectorSection.Command, Original_Input = data }); //C7 //Sets to Read //(0 1 1: Read)
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C6", Section =
VectorSection.Command, Original_Input = data }); //C6 //Sets to Read
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C5", Section =
VectorSection.Command, Original_Input = data }); //C5 //Sets to Read

for (int i = 3; i < 8; i++) //reg address: C4, C3, C2, C1, C0
{
command.Add(new Operation(timeSet, 1.ToString(), rRegisterBinary[i].ToString()) { Comment =
"//C" + (rRegisterBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input = data
});
}
//Append slave address bits and write/read bits (first 3 C7,6,5 bits) then Cut off the first 3 bits of
registerBinary before calculating the parity bit
string rCommadFrameBinary = rUserIdBinary + readCommandValues +
rRegisterBinary.Substring(3); //registerBinary.Substring(3) gets rid of the first 3 bit/chars to allow
space for the command values in the 8 bit commad pay load.
int rCommandParityBit = rCommadFrameBinary.Count(c => c == '1') % 2; //command
string rCommandParityBitString = rCommandParityBit == 0 ? "1" : "0"; //command
command.Add(new Operation(timeSet, 1.ToString(), rCommandParityBitString) { Comment =
"//P", Section = VectorSection.Command, Original_Input = data }); //Add in the parity bit.

// Create a sequence of operations for the data
for (int i = 0; i < 8; i++) //data: D7, D6, D5, D4, D3, D2, D1, D0
{
dataSequence.Add(new Operation(timeSet, 1.ToString(), rDataBinary[i].ToString()) { Comment =
"//D" + (rDataBinary.Length - (i + 1)), Section = VectorSection.Data, Original_Input = data });
}
// Calculate the parity bit for the data and command
int rDataParityBit = rDataBinary.Count(c => c == '1') % 2; //data
string rDataParityBitString = rDataParityBit == 0 ? "1" : "0"; //data
dataSequence.Add(new Operation(timeSet, 1.ToString(), rDataParityBitString) { Comment = "//P",
```

```
Section = VectorSection.Data, Original_Input = data }); //Write in the parity bit.
#endregion
break;
case RffePatternSpec.ZW:
#region Zero Write

// Convert the register and data values to binary strings according to pattern spec / bit
requirements.
string zwUserIdBinary = HexToBinary(UserIDValue);
//string registerBinary = HexToBinary(register);
string zwDataBinary = HexToBinary(data);

//Dynamic slave address
for (int i = 0; i < 4; i++) //reg address: SA3, SA2, SA1, SA0
{
command.Add(new Operation(timeSet, 1.ToString(), zwUserIdBinary[i].ToString()) { Comment =
"//SA" + (zwUserIdBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input =
data });
}

// Create a static, spec specific, sequence of operations for the command.
string zeroWriteCommandValues = "0"; //As seen below. (Non-dynamic initial sequence)
//Zero Write (0)
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//ZW", Section =
VectorSection.Command, Original_Input = data }); //C7 //Sets to Zero Write

// Create a sequence of operations for the data to fill the command frame.
for (int i = 1; i < 8; i++) //data: D6, D5, D4, D3, D2, D1, D0
{
command.Add(new Operation(timeSet, 1.ToString(), zwDataBinary[i].ToString()) { Comment =
"//D" + (zwDataBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input = data
});
}
//Append slave address bits and zero write bits (first bit C7) then Cut off the first bit of
zwDataBinary before calculating the parity bit
string zwCommadFrameBinary = zwUserIdBinary + zeroWriteCommandValues +
zwDataBinary.Substring(1); //zwDataBinary.Substring(1) gets rid of the first 1 bit/chars to allow
space for the command values in the 8 bit commad pay load.
int zwCommandParityBit = zwDataBinary.Count(c => c == '1') % 2; //command
string zwCommandParityBitString = zwCommandParityBit == 0 ? "1" : "0"; //command
command.Add(new Operation(timeSet, 1.ToString(), zwCommandParityBitString) { Comment =
"//P", Section = VectorSection.Command, Original_Input = data }); //Add in the parity bit.
```

```
#endregion
break;
case RffePatternSpec.MW:
#region Masked Write

// Convert the register and data values to binary strings according to pattern spec / bit requirements.
string mwUserIdBinary = HexToBinary(this.UserIDValue);
string mwRegisterBinary = HexToBinary(register);
string mwMaskBinary = HexToBinary(this.MaskValue);
string mwDataBinary = HexToBinary(data); //HexToBinary(data);
//for (int i = 2; i < data.Length; i++)
//{
//    mwDataBinary += HexToBinary(data[i].ToString());
//}

//Dynamic slave address
for (int i = 0; i < 4; i++) //reg address: SA3, SA2, SA1, SA0
{
command.Add(new Operation(timeSet, 1.ToString(), mwUserIdBinary[i].ToString()) { Comment = "//SA" + (mwUserIdBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input = data });
}

// Create a static, spec specific, sequence of operations for the command.
string maskedWriteCommandValues = "0001 1001 "; //As seen below. (Non-dynamic initial sequence)
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C7", Section = VectorSection.Command, Original_Input = data }); //C7
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C6", Section = VectorSection.Command, Original_Input = data }); //C6
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C5", Section = VectorSection.Command, Original_Input = data }); //C5
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C4", Section = VectorSection.Command, Original_Input = data }); //C4

command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C3", Section = VectorSection.Command, Original_Input = data }); //C3
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C2", Section = VectorSection.Command, Original_Input = data }); //C2
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C1", Section = VectorSection.Command, Original_Input = data }); //C1
```

```
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C0", Section =
VectorSection.Command, Original_Input = data }); //C0

//Append slave address bits and write/read bits (first 3 C7,6,5 bits) then Cut off the first 3 bits of
registerBinary before calculating the parity bit
string mwCommadFrameBinary = mwUserIdBinary + maskedWriteCommandValues;
int mwCommandParityBit = mwCommadFrameBinary.Count(c => c == '1') % 2; //command
string mwCommandParityBitString = mwCommandParityBit == 0 ? "1" : "0"; //command
command.Add(new Operation(timeSet, 1.ToString(), mwCommandParityBitString) { Comment =
"//Parity", Section = VectorSection.Command, Original_Input = data }); //Add in the parity bit.

for (int i = 0; i < 8; i++) //reg address: C7, C6, C5, C4, C3, C2, C1, C0
{
registerSequence.Add(new Operation(timeSet, 1.ToString(), mwRegisterBinary[i].ToString()) {
Comment = "//A" + (mwRegisterBinary.Length - (i + 1)), Section =
VectorSection.Register_Address, Original_Input = data });
}
string registerAddressBinary = mwRegisterBinary; //8 bit
int registerAddressParityBit = registerAddressBinary.Count(c => c == '1') % 2; //command
string registerAddressParityBitString = registerAddressParityBit == 0 ? "1" : "0"; //command
registerSequence.Add(new Operation(timeSet, 1.ToString(), registerAddressParityBitString) {
Comment = "//Parity", Section = VectorSection.Register_Address, Original_Input = data }); //Add in
the parity bit.

for (int i = 0; i < 8; i++) //mask 8 bit
{
maskEightBit.Add(new Operation(timeSet, 1.ToString(), mwMaskBinary[i].ToString()) { Comment =
"//M" + (mwMaskBinary.Length - (i + 1)), Section = VectorSection.Mask_EightBit, Original_Input =
data });
}
string maskBinary = mwMaskBinary; //8 bit
int maskParityBit = maskBinary.Count(c => c == '1') % 2; //command
string maskParityBitString = maskParityBit == 0 ? "1" : "0"; //command
maskEightBit.Add(new Operation(timeSet, 1.ToString(), maskParityBitString) { Comment =
"//Parity", Section = VectorSection.Mask_EightBit, Original_Input = data }); //Add in the parity bit.

// Create a sequence of operations for the data
for (int i = 0; i < 8; i++) //data: D7, D6, D5, D4, D3, D2, D1, D0
{
dataSequence.Add(new Operation(timeSet, 1.ToString(), mwDataBinary[i].ToString()) { Comment
= "//D" + (mwDataBinary.Length - (i + 1)), Section = VectorSection.Data, Original_Input = data });
}
```

```
// Calculate the parity bit for the data and command
int mwDataParityBit = mwDataBinary.Count(c => c == '1') % 2; //data
string mwDataParityBitString = mwDataParityBit == 0 ? "1" : "0"; //data
dataSequence.Add(new Operation(timeSet, 1.ToString(), mwDataParityBitString) { Comment =
"//Parity", Section = VectorSection.Data, Original_Input = data }); //Write in the parity bit.
#endregion
break;

case RffePatternSpec.EW:
#region Extended Write
// Convert the register and data values to binary strings according to pattern spec / bit
requirements.
string ewUserIdBinary = HexToBinary(this.UserIDValue);
string ewRegisterBinary = HexToBinary(register);

//Dynamic slave address
for (int i = 0; i < 4; i++) //reg address: SA3, SA2, SA1, SA0
{
command.Add(new Operation(timeSet, 1.ToString(), ewUserIdBinary[i].ToString()) { Comment =
"//SA" + (ewUserIdBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input =
data });
}

// Create a static, spec specific, sequence of operations for the command.
string extendedWriteCommandValues = "0000"; //As seen below. (Non-dynamic initial sequence)
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C7", Section =
VectorSection.Command, Original_Input = data }); //C7
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C6", Section =
VectorSection.Command, Original_Input = data }); //C6
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C5", Section =
VectorSection.Command, Original_Input = data }); //C5
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C4", Section =
VectorSection.Command, Original_Input = data }); //C4

//Byte Count? Calculated
for (int i = 0; i < this.ByteCountBinary.Length; i++)
{
command.Add(new Operation(timeSet, 1.ToString(), this.ByteCountBinary[i].ToString()) {
Comment = "//C" + (this.ByteCountBinary.Length - (i + 1)), Section = VectorSection.Command,
Original_Input = data }); //C3
}
```

```
//Append slave address bits and write/read bits (first 3 C7,6,5 bits) then Cut off the first 3 bits of registerBinary before calculating the parity bit
string ewCommadFrameBinary = ewUserIdBinary + extendedWriteCommandValues + this.ByteCountBinary;
int ewCommandParityBit = ewCommadFrameBinary.Count(c => c == '1') % 2; //command
string ewCommandParityBitString = ewCommandParityBit == 0 ? "1" : "0"; //command
command.Add(new Operation(timeSet, 1.ToString(), ewCommandParityBitString) { Comment = "//Parity", Section = VectorSection.Command, Original_Input = data }); //Add in the parity bit.

for (int i = 0; i < 8; i++) //reg address: C7, C6, C5, C4, C3, C2, C1, C0
{
registerSequence.Add(new Operation(timeSet, 1.ToString(), ewRegisterBinary[i].ToString()) { Comment = "//A" + (ewRegisterBinary.Length - (i + 1)), Section = VectorSection.Register_Address, Original_Input = data });
}
string ewRegisterAddressBinary = ewRegisterBinary; //8 bit
int ewRegisterAddressParityBit = ewRegisterAddressBinary.Count(c => c == '1') % 2; //command
string ewRegisterAddressParityBitString = ewRegisterAddressParityBit == 0 ? "1" : "0"; //command
registerSequence.Add(new Operation(timeSet, 1.ToString(), ewRegisterAddressParityBitString) { Comment = "//Parity", Section = VectorSection.Register_Address, Original_Input = data }); //Add in the parity bit.

//Create a sequence of operations for the data.
for (int strIdx = 0; strIdx < data.Length; strIdx += 2)
{
string twoCharSubstring = data.Substring(strIdx, 2);
string ewDataBinary = HexToBinary(twoCharSubstring); //HexToBinary(data);

for (int i = 0; i < 8; i++) // data: D7, D6, D5, D4, D3, D2, D1, D0
{
dataSequence.Add(new Operation(timeSet, 1.ToString(), ewDataBinary[i].ToString()) { Comment = "//D" + (ewDataBinary.Length - (i + 1)), Section = VectorSection.Data, Original_Input = data });
}

// Calculate the parity bit for the data and command
int ewDataParityBit = ewDataBinary.Count(c => c == '1') % 2; // data
string ewDataParityBitString = ewDataParityBit == 0 ? "1" : "0"; // data
dataSequence.Add(new Operation(timeSet, 1.ToString(), ewDataParityBitString) { Comment = "//Parity", Section = VectorSection.Data, Original_Input = data }); // Write in the parity bit.
}
#endregion
break;
```

```
case RffePatternSpec.ER:
#region Extended Read
// Convert the register and data values to binary strings according to pattern spec / bit requirements.
string erUserIdBinary = HexToBinary(this.UserIDValue);
string erRegisterBinary = HexToBinary(register);

//Dynamic slave address
for (int i = 0; i < 4; i++) //reg address: SA3, SA2, SA1, SA0
{
command.Add(new Operation(timeSet, 1.ToString(), erUserIdBinary[i].ToString()) { Comment = "//SA" + (erUserIdBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input = data });
}

// Create a static, spec specific, sequence of operations for the command.
string extendedReadCommandValues = "0010"; //As seen below. (Non-dynamic initial sequence)
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C7", Section = VectorSection.Command, Original_Input = data }); //C7
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C6", Section = VectorSection.Command, Original_Input = data }); //C6
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C5", Section = VectorSection.Command, Original_Input = data }); //C5
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C4", Section = VectorSection.Command, Original_Input = data }); //C4

//Byte Count? Calculated
for (int i = 0; i < this.ByteCountBinary.Length; i++)
{
command.Add(new Operation(timeSet, 1.ToString(), this.ByteCountBinary[i].ToString()) { Comment = "//C" + (this.ByteCountBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input = data }); //C3
}

//Append slave address bits and write/read bits (first 3 C7,6,5 bits) then Cut off the first 3 bits of registerBinary before calculating the parity bit
string erCommadFrameBinary = erUserIdBinary + extendedReadCommandValues + this.ByteCountBinary;
int erCommandParityBit = erCommadFrameBinary.Count(c => c == '1') % 2; //command
string erCommandParityBitString = erCommandParityBit == 0 ? "1" : "0"; //command
command.Add(new Operation(timeSet, 1.ToString(), erCommandParityBitString) { Comment = "//Parity", Section = VectorSection.Command, Original_Input = data }); //Add in the parity bit.
```

```
for (int i = 0; i < 8; i++) //reg address: C7, C6, C5, C4, C3, C2, C1, C0
{
registerSequence.Add(new Operation(timeSet, 1.ToString(), erRegisterBinary[i].ToString()) {
Comment = "//A" + (erRegisterBinary.Length - (i + 1)), Section = VectorSection.Register_Address,
Original_Input = data });
}
string erRegisterAddressBinary = erRegisterBinary; //8 bit
int erRegisterAddressParityBit = erRegisterAddressBinary.Count(c => c == '1') % 2; //command
string erRegisterAddressParityBitString = erRegisterAddressParityBit == 0 ? "1" : "0"; //command
registerSequence.Add(new Operation(timeSet, 1.ToString(), erRegisterAddressParityBitString) {
Comment = "//Parity", Section = VectorSection.Register_Address, Original_Input = data }); //Add in
the parity bit.

//Create a sequence of operations for the data.
for (int strIdx = 0; strIdx < data.Length; strIdx += 2)
{
string twoCharSubstring = data.Substring(strIdx, 2);
string erDataBinary = HexToBinary(twoCharSubstring); //HexToBinary(data);

for (int i = 0; i < 8; i++) // data: D7, D6, D5, D4, D3, D2, D1, D0
{
dataSequence.Add(new Operation(timeSet, 1.ToString(), erDataBinary[i].ToString()) { Comment =
"//D" + (erDataBinary.Length - (i + 1)), Section = VectorSection.Data, Original_Input = data });
}

// Calculate the parity bit for the data and command
int erDataParityBit = erDataBinary.Count(c => c == '1') % 2; // data
string erDataParityBitString = erDataParityBit == 0 ? "1" : "0"; // data
dataSequence.Add(new Operation(timeSet, 1.ToString(), erDataParityBitString) { Comment =
"//Parity", Section = VectorSection.Data, Original_Input = data }); // Write in the parity bit.
}
#endregion
break;

case RffePatternSpec.LEW:
#region Extended Write
// Convert the register and data values to binary strings according to pattern spec / bit
requirements.
string lewUserIdBinary = HexToBinary(this.UserIDValue);
//string lewRegisterBinary = HexToBinary(register);
```

```
//Dynamic slave address
for (int i = 0; i < 4; i++) //reg address: SA3, SA2, SA1, SA0
{
command.Add(new Operation(timeSet, 1.ToString(), lewUserIdBinary[i].ToString()) { Comment =
"//SA" + (lewUserIdBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input =
data });
}

// Create a static, spec specific, sequence of operations for the command.
string longExtendedWriteCommandValues = "00110"; //As seen below. (Non-dynamic initial
sequence)
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C7", Section =
VectorSection.Command, Original_Input = data }); //C7
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C6", Section =
VectorSection.Command, Original_Input = data }); //C6
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C5", Section =
VectorSection.Command, Original_Input = data }); //C5
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C4", Section =
VectorSection.Command, Original_Input = data }); //C4
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C4", Section =
VectorSection.Command, Original_Input = data }); //C3

//Byte Count? Calculated (3 bits)
for (int i = 0; i < this.ByteCountBinary.Length; i++)
{
command.Add(new Operation(timeSet, 1.ToString(), this.ByteCountBinary[i].ToString()) {
Comment = "//C" + (this.ByteCountBinary.Length - (i + 1)), Section = VectorSection.Command,
Original_Input = data }); //C3
}

//Append slave address bits and write/read bits (first 3 C7,6,5 bits) then Cut off the first 3 bits of
registerBinary before calculating the parity bit
string lewCommadFrameBinary = lewUserIdBinary + longExtendedWriteCommandValues +
this.ByteCountBinary;
int lewCommandParityBit = lewCommadFrameBinary.Count(c => c == '1') % 2; //command
string lewCommandParityBitString = lewCommandParityBit == 0 ? "1" : "0"; //command
command.Add(new Operation(timeSet, 1.ToString(), lewCommandParityBitString) { Comment =
"//Parity", Section = VectorSection.Command, Original_Input = data }); //Add in the parity bit.

for (int strIdx = 0; strIdx < register.Length; strIdx += 2)
{
string twoCharSubstring = register.Substring(strIdx, 2);
```

```
string lewRegisterBinary = HexToBinary(twoCharSubstring); //HexToBinary(data);

for (int i = 0; i < 8; i++) //reg address: C7, C6, C5, C4, C3, C2, C1, C0
{
registerSequence.Add(new Operation(timeSet, 1.ToString(), lewRegisterBinary[i].ToString()) {
Comment = "//A" + (lewRegisterBinary.Length - (i + 1)), Section =
VectorSection.Register_Address, Original_Input = data });
}
string lewRegisterAddressBinary = lewRegisterBinary; //8 bit
int lewRegisterAddressParityBit = lewRegisterAddressBinary.Count(c => c == '1') % 2; //command
string lewRegisterAddressParityBitString = lewRegisterAddressParityBit == 0 ? "1" : "0";
//command
registerSequence.Add(new Operation(timeSet, 1.ToString(), lewRegisterAddressParityBitString) {
Comment = "//Parity", Section = VectorSection.Register_Address, Original_Input = data }); //Add in
the parity bit.
}

//Create a sequence of operations for the data.
for (int strIdx = 0; strIdx < data.Length; strIdx += 2)
{
string twoCharSubstring = data.Substring(strIdx, 2);
string lewDataBinary = HexToBinary(twoCharSubstring); //HexToBinary(data);

for (int i = 0; i < 8; i++) // data: D7, D6, D5, D4, D3, D2, D1, D0
{
dataSequence.Add(new Operation(timeSet, 1.ToString(), lewDataBinary[i].ToString()) { Comment
= "//D" + (lewDataBinary.Length - (i + 1)), Section = VectorSection.Data, Original_Input = data });
}

// Calculate the parity bit for the data and command
int lewDataParityBit = lewDataBinary.Count(c => c == '1') % 2; // data
string lewDataParityBitString = lewDataParityBit == 0 ? "1" : "0"; // data
dataSequence.Add(new Operation(timeSet, 1.ToString(), lewDataParityBitString) { Comment =
"//Parity", Section = VectorSection.Data, Original_Input = data }); // Write in the parity bit.
}
#endregion
break;
case RffePatternSpec.LER:
#region Extended Write
// Convert the register and data values to binary strings according to pattern spec / bit
requirements.
string lerUserIdBinary = HexToBinary(this.UserIDValue);
```

```
//string lewRegisterBinary = HexToBinary(register);

//Dynamic slave address
for (int i = 0; i < 4; i++) //reg address: SA3, SA2, SA1, SA0
{
command.Add(new Operation(timeSet, 1.ToString(), lerUserIdBinary[i].ToString()) { Comment =
"//SA" + (lerUserIdBinary.Length - (i + 1)), Section = VectorSection.Command, Original_Input =
data });
}

// Create a static, spec specific, sequence of operations for the command.
string longExtendedReadCommandValues = "00111"; //As seen below. (Non-dynamic initial
sequence)
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C7", Section =
VectorSection.Command, Original_Input = data }); //C7
command.Add(new Operation(timeSet, 1.ToString(), "0") { Comment = "//C6", Section =
VectorSection.Command, Original_Input = data }); //C6
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C5", Section =
VectorSection.Command, Original_Input = data }); //C5
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C4", Section =
VectorSection.Command, Original_Input = data }); //C4
command.Add(new Operation(timeSet, 1.ToString(), "1") { Comment = "//C4", Section =
VectorSection.Command, Original_Input = data }); //C3

//Byte Count? Calculated (3 bits)
for (int i = 0; i < this.ByteCountBinary.Length; i++)
{
command.Add(new Operation(timeSet, 1.ToString(), this.ByteCountBinary[i].ToString()) {
Comment = "//C" + (this.ByteCountBinary.Length - (i + 1)), Section = VectorSection.Command,
Original_Input = data }); //C3
}

//Append slave address bits and write/read bits (first 3 C7,6,5 bits) then Cut off the first 3 bits of
registerBinary before calculating the parity bit
string lerCommadFrameBinary = lerUserIdBinary + longExtendedReadCommandValues +
this.ByteCountBinary;
int lerCommandParityBit = lerCommadFrameBinary.Count(c => c == '1') % 2; //command
string lerCommandParityBitString = lerCommandParityBit == 0 ? "1" : "0"; //command
command.Add(new Operation(timeSet, 1.ToString(), lerCommandParityBitString) { Comment =
"//Parity", Section = VectorSection.Command, Original_Input = data }); //Add in the parity bit.

for (int strIdx = 0; strIdx < register.Length; strIdx += 2)
```

```
{
string twoCharSubstring = register.Substring(strIdx, 2);
string lerRegisterBinary = HexToBinary(twoCharSubstring); //HexToBinary(data);

for (int i = 0; i < 8; i++) //reg address: C7, C6, C5, C4, C3, C2, C1, C0
{
registerSequence.Add(new Operation(timeSet, 1.ToString(), lerRegisterBinary[i].ToString()) {
Comment = "//A" + (lerRegisterBinary.Length - (i + 1)), Section = VectorSection.Register_Address,
Original_Input = data });
}
string lerRegisterAddressBinary = lerRegisterBinary; //8 bit
int lerRegisterAddressParityBit = lerRegisterAddressBinary.Count(c => c == '1') % 2; //command
string lerRegisterAddressParityBitString = lerRegisterAddressParityBit == 0 ? "1" : "0"; //command
registerSequence.Add(new Operation(timeSet, 1.ToString(), lerRegisterAddressParityBitString) {
Comment = "//Parity", Section = VectorSection.Register_Address, Original_Input = data }); //Add in
the parity bit.
}

//Create a sequence of operations for the data.
for (int strIdx = 0; strIdx < data.Length; strIdx += 2)
{
string twoCharSubstring = data.Substring(strIdx, 2);
string lerDataBinary = HexToBinary(twoCharSubstring); //HexToBinary(data);

for (int i = 0; i < 8; i++) // data: D7, D6, D5, D4, D3, D2, D1, D0
{
dataSequence.Add(new Operation(timeSet, 1.ToString(), lerDataBinary[i].ToString()) { Comment =
"//D" + (lerDataBinary.Length - (i + 1)), Section = VectorSection.Data, Original_Input = data });
}

// Calculate the parity bit for the data and command
int lerDataParityBit = lerDataBinary.Count(c => c == '1') % 2; // data
string lerDataParityBitString = lerDataParityBit == 0 ? "1" : "0"; // data
dataSequence.Add(new Operation(timeSet, 1.ToString(), lerDataParityBitString) { Comment =
"//Parity", Section = VectorSection.Data, Original_Input = data }); // Write in the parity bit.
}
#endregion
break;

default:
//Maybe should be Write (W) but for now do nothing.
break;
```

```
}
///DYNAMIC

// Create a static sequence of operations for the bus park
List<Operation> busPark = new List<Operation>()
{
new Operation(timeSet, 1.ToString(), "0") { Section = VectorSection.Bus_Park, Comment =
"//BPC", Original_Input = data },
new Operation(timeSet, 0.ToString(), "X") { Section = VectorSection.Bus_Park, Comment =
"//BPC", Original_Input = data },
new Operation(timeSet, "0", "0") { Section = VectorSection.Bus_Park, Comment = "//BPC",
Original_Input = data }
};

//combine all operations to master list!
MasterOps.AddRange(startSequence);
MasterOps.AddRange(command);
MasterOps.AddRange(registerSequence);
MasterOps.AddRange(maskEightBit);
MasterOps.AddRange(dataSequence);
MasterOps.AddRange(busPark);

//Apply vector numbers to all the operations in master ops.
for (int i = 0; i < MasterOps.Count; i++)
{
MasterOps[i].VectorNumber = i + 1;
}
}
catch(Exception ex)
{
//MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
}
return new ObservableCollection<Operation>(MasterOps);
}

public static KeyValuePair<RffePatternSpec, string> ContainsAnyRffePatternSpec(string input)
{
foreach (var kvp in BackendConstants.RffePatternSpecKvPs)
{
if (input.Contains(kvp.Value))
```

```
{
return kvp;
}
}
return BackendConstants.RffePatternSpecKvPs[0]; //Return Regular Write As default (MAY CAUSE BUGS);
}

//NEEDS TO SECTION OUT THE UNDERSCORE SEPARATED DATA
//For removing the RffePatternSpec characters if it's included in the data input.
public static string RemoveNonHexCharacters(string input)
{
// Use a regular expression to match non-hex characters
var regex = new Regex("[^0-9a-fA-F]+");

// Remove non-hex characters and return the result
return regex.Replace(input, "");
}

public static string HexToBinary(string hexString)
{
string binaryString = string.Empty;

foreach (char hexChar in hexString)
{
int hexValue = Convert.ToInt32(hexChar.ToString(), 16);
string binaryValue = Convert.ToString(hexValue, 2).PadLeft(4, '0');
binaryString += binaryValue;
}

return binaryString;
}

public static string[] SeparateStringByUnderscore(string input)
{
if (string.IsNullOrEmpty(input))
{
return new string[0];
}

return input.Split('_');
}
```

```
private bool IsValidBinary(string input)
{
return Regex.IsMatch(input, "^[01]+$");
}
#endregion

/// <summary>
/// Checks the cal point validation status.
/// </summary>
private void CheckPointStatus()
{
List<ValidationResult> statusResults = new List<ValidationResult>();

//BUG: For some reason it allows the letter "a" through and breaks the entire process. (Caps letters only ??)
//Checks if the data only contains only C-style hex notation (0xFF) values
if (!DigitalPatternModel.OnlyHexInString(this.InputValue))
{
//Console.WriteLine($"RFFE input invalid: {this.InputValue}, invalid hex data for Register {this.RegisterValue}");
ValidationResult SourcePortResult = new ValidationResult()
{
PropertyName = "Invalid Data",
ErrorMessage = "The Data must be hex characters only"
};
statusResults.Add(SourcePortResult);
}
//BUG: For some reason it allows the letter "a" through and breaks the entire process. (Caps letters only ??)
//Checks if the data only contains only C-style hex notation (0xFF) values
if (!DigitalPatternModel.OnlyHexInString(this.RegisterValue))
{
ValidationResult SourcePortResult = new ValidationResult()
{
PropertyName = "Invalid Register",
ErrorMessage = "The Register must be hex characters only"
};
statusResults.Add(SourcePortResult);
}

//BUG: For some reason it allows the letter "a" through and breaks the entire process. (Caps
```

```
letters only ??)
//Checks if the data only contains only C-style hex notation (0xFF) values
if (!DigitalPatternModel.OnlyHexInString(this.UserIDValue))
{
ValidationResult SourcePortResult = new ValidationResult()
{
PropertyName = "Invalid User ID",
ErrorMessage = "The User ID must be hex characters only"
};
statusResults.Add(SourcePortResult);
}
//Check UserIDValue is 4 bits or 1 hex character in the string exactly
if (this.UserIDValue.Length != 1)
{
ValidationResult maskLengthResult = new ValidationResult()
{
PropertyName = "Invalid User ID",
ErrorMessage = "The User ID must be exactly 4 bits in length."
};
statusResults.Add(maskLengthResult);
}

switch (this.Spec)
{
case RffePatternSpec.W:
case RffePatternSpec.R:
// Check that the RegisterValue property has a valid length (exactly 1 byte)
if (this.RegisterValue.Length != 2)
{
ValidationResult registerLengthResult = new ValidationResult()
{
PropertyName = "Invalid Register",
ErrorMessage = "The Register property must be exactly 1 byte in length."
};
statusResults.Add(registerLengthResult);
}

// Check that the InputValue property has a valid length (no longer than 8 bits or 1 byte)
if (this.InputValue.Length > 2)
{
ValidationResult inputLengthResult = new ValidationResult()
{
```

```
PropertyName = "Invalid Data",
ErrorMessage = "The Data property must be no longer than 8 bits or 1 byte in length."
};
statusResults.Add(inputLengthResult);
}
break;

case RffePatternSpec.ZW:
// Check that the InputValue property has a valid length (no longer than 8 bits or 1 byte)
if (this.InputValue.Length > 2)
{
ValidationResult inputLengthResult = new ValidationResult()
{
PropertyName = "Invalid Data",
ErrorMessage = "The Data property must be no longer than 8 bits or 1 byte in length."
};
statusResults.Add(inputLengthResult);
}
break;

case RffePatternSpec.MW:
// Check that the MaskValue property only contains valid hexadecimal characters
if (!DigitalPatternModel.OnlyHexInString(this.MaskValue))
{
ValidationResult maskResult = new ValidationResult()
{
PropertyName = "Invalid Mask",
ErrorMessage = "The Mask Value property must be a valid hexadecimal string."
};
statusResults.Add(maskResult);
}

// Check that the MaskValue property has a valid length (exactly 1 byte)
if (this.MaskValue.Length != 2)
{
ValidationResult maskLengthResult = new ValidationResult()
{
PropertyName = "Invalid Mask",
ErrorMessage = "The Mask Value must be exactly 1 byte in length."
};
statusResults.Add(maskLengthResult);
}
```

```
break;

case RffePatternSpec.EW:
case RffePatternSpec.ER:
// Check that the ByteCountBinary property only contains valid binary characters
if (!IsValidBinary(this.ByteCountBinary))
{
ValidationResult byteCountResult = new ValidationResult()
{
PropertyName = "Invalid Byte Count",
ErrorMessage = "The Byte Count property must be a valid binary string."
};
statusResults.Add(byteCountResult);
}
// Check that the ByteCountBinary property is only 3 bits in length
if (this.ByteCountBinary.Length != 4)
{
ValidationResult registerLengthResult = new ValidationResult()
{
PropertyName = "Invalid Byte Count",
ErrorMessage = "The Byte Count property must be exactly 4 bits in length."
};
statusResults.Add(registerLengthResult);
}

// Check that the RegisterValue property has a valid length (exactly 1 byte)
if (this.RegisterValue.Length != 2)
{
ValidationResult registerLengthResult = new ValidationResult()
{
PropertyName = "Invalid Register",
ErrorMessage = "The Register property must be exactly 1 byte in length."
};
statusResults.Add(registerLengthResult);
}

// Check that the InputValue property has a valid length (no shorter than 1 byte and no longer than
8 bytes)
if (this.InputValue.Length < 2 || this.InputValue.Length > 16)
{
ValidationResult inputLengthResult = new ValidationResult()
{
```

```
PropertyName = "Invalid Data",
ErrorMessage = "The Data property must be at least 1 byte and no longer than 8 bytes in length."
};
statusResults.Add(inputLengthResult);
}

break;

case RffePatternSpec.LEW:
case RffePatternSpec.LER:
// Check that the ByteCountBinary property only contains valid binary characters
if (!IsValidBinary(this.ByteCountBinary))
{
ValidationResult byteCountResult = new ValidationResult()
{
PropertyName = "ByteCountBinary",
ErrorMessage = "The Byte Count property must be a valid binary string."
};
statusResults.Add(byteCountResult);
}
// Check that the ByteCountBinary property is only 3 bits in length
if (this.ByteCountBinary.Length != 3)
{
ValidationResult registerLengthResult = new ValidationResult()
{
PropertyName = "ByteCountBinary",
ErrorMessage = "The Byte Count property must be exactly 3 bits in length."
};
statusResults.Add(registerLengthResult);
}

// Check that the RegisterValue property has a valid length (exactly 2 bytes)
if (this.RegisterValue.Length != 4)
{
ValidationResult registerLengthResult = new ValidationResult()
{
PropertyName = "RegisterValue",
ErrorMessage = "The Register property must be exactly 2 bytes in length."
};
statusResults.Add(registerLengthResult);
}
```

```
// Check that the InputValue property has a valid length (no shorter than 1 byte and no longer than 16 bytes)
if (this.InputValue.Length < 2 || this.InputValue.Length > 32)
{
ValidationResult inputLengthResult = new ValidationResult()
{
PropertyName = "InputValue",
ErrorMessage = "The Data property must be at least 1 byte and no longer than 16 bytes in length."
};
statusResults.Add(inputLengthResult);
}

break;

default:
break;
}



//Error message handling logic goes here.
if (statusResults.Count() > 0)
{
Status = ValidationStatus.Invalid;
ErrorMessage = string.Empty; //Clear ToolTip
foreach (ValidationResult vr in statusResults) //Re-compile ToolTip notifications.
{
if (!string.IsNullOrEmpty(ErrorMessage)) { ErrorMessage += Environment.NewLine; }
ErrorMessage += string.Format("{0}: {1}.", vr.PropertyName, vr.ErrorMessage);
}
//ValidationStatusChanged?.Invoke(this);
}
else
{
Status = ValidationStatus.Valid;
ErrorMessage = string.Empty;
//ValidationStatusChanged?.Invoke(this);
}

}

#region INPC Members
[field: NonSerialized]
```

```
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
#endregion
}

[Serializable]
public class GenericRffePatternModel : INotifyPropertyChanged, IFileData
{
#region Private Members
private string _filePath;
private DigiTimeObj _timeSetUsedObj;
private string _timeSetItem;
private PinModel _clockPin;
private string _clockPinItem;
private PinModel _dataPin;
private string _dataPinItem;
private ObservableCollection<GenericMode> _modeCollection = new
ObservableCollection<GenericMode>();
private ObservableCollection<string> _registers = new ObservableCollection<string>();

private bool _isCondensed = false;
#endregion

#region Public Members
public string FileName => Path.GetFileName(this.FilePath);
public string FilePath
{
get { return _filePath; }
set { _filePath = value; OnPropertyChanged("FilePath"); OnPropertyChanged("FileName"); }
}



public string TimeSetUsed
{
get { return _timeSetItem; }
set { _timeSetItem = value; OnPropertyChanged("TimeSetUsed"); }
}
[XmlIgnore]
public DigiTimeObj TimeSetUsedObj
```

```
{
get { return _timeSetUsedObj; }
set
{
_timeSetUsedObj = value; OnPropertyChanged("TimeSetUsedObj");
if (value != null)
{
TimeSetUsed = value.Name;
foreach (GenericMode m in ModeCollection)
{
foreach (GenericModeVector mv in m.ModeVectors)
{
foreach (Operation op in mv.VectorizedData)
{
op.TimeSet = value.Name;
}
}
}
this.CompileAllModes();
}
}
}

public string ClockPinItem
{
get { return _clockPinItem; }
set { _clockPinItem = value; OnPropertyChanged("ClockPinItem"); }
}
[XmlIgnore]
public PinModel ClockPin
{
get { return _clockPin; }
set
{
_clockPin = value;  OnPropertyChanged("ClockPin");
if (value != null)
{
ClockPinItem = value.PinName;
this.CompileAllModes();
}
}
}
```

```
public string DataPinItem
{
get { return _dataPinItem; }
set { _dataPinItem = value; OnPropertyChanged("DataPinItem"); }
}
[XmlIgnore]
public PinModel DataPin
{
get { return _dataPin; }
set
{
_dataPin = value; OnPropertyChanged("DataPin");
if (value != null)
{
DataPinItem = value.PinName;
this.CompileAllModes();
}
}
}

//Columns
public ObservableCollection<string> Registers
{
get { return _registers; }
set { _registers = value; OnPropertyChanged("Registers"); }
}

public bool IsCondensed
{
get { return _isCondensed; }
set { _isCondensed = value; OnPropertyChanged("IsCondensed"); }
}

//Rows
public ObservableCollection<GenericMode> ModeCollection
{
get { return _modeCollection; }
set { _modeCollection = value; OnPropertyChanged("ModeCollection"); }
}

#endregion
```

```
#region Events
public event Action ChangeOccured;
public event Action CommitEditBeforeSave;
#endregion

#region Constructor
public GenericRffePatternModel() { } //Deserialization Constructor
public GenericRffePatternModel(string filePath)
{
this.FilePath = filePath;

//Temp.
ModeCollection.Add(new GenericMode() { Name = "Mode Name Here", ModeVectors = new
ObservableCollection<GenericModeVector>() { new GenericModeVector("00") } });
}
#endregion

public void Save()
{
CommitEditBeforeSave?.Invoke();
this.SaveToXml(this.FilePath);
Console.WriteLine(string.Format("Saved: {0}", this.FilePath));
}
public void Load()
{

}
public bool Undo()
{
Console.WriteLine("Undo operation not implemented for Digital Timing Files");
return true;
}

public bool Redo()
{
Console.WriteLine("Undo operation not implemented for Digital Timing Files");
return true;
}

#region XML Import/Export
```

```
public void SaveToXml(string FileName)
{
try
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(FileName))
{
xmlSerializer.Serialize(writer, this);
}
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
}
public static GenericRffePatternModel LoadFromXml(string FileName)
{
try
{
XmlSerializer ser = new XmlSerializer(typeof(GenericRffePatternModel));
GenericRffePatternModel digiPatternModelAfterDeSerialize;

using (StreamReader sr = new StreamReader(FileName))// Deserialize XML file.
{
digiPatternModelAfterDeSerialize = (GenericRffePatternModel)ser.Deserialize(sr);
}

return digiPatternModelAfterDeSerialize;
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); return null; }
}
#endregion
public void RelinkPinData(PinMapModel projectPinMap)
{
foreach (Data.PinModel pin in projectPinMap.DigitalPins.Where(pin => pin.PinName ==
DataPinItem))
{
DataPin = pin;
}
foreach (Data.PinModel pin in projectPinMap.DigitalPins.Where(pin => pin.PinName ==
ClockPinItem))
{
ClockPin = pin;
```

```
}
}


public void GeneratePatternFiles(string FileName, string pinMapFilePath)
{
//Create a sub directory or make sure it exists to store all the generated pattern files.
string dir = Path.GetDirectoryName(FileName);
string folder = Path.GetFileNameWithoutExtension(FileName);
string OutputDir = Path.Combine(dir, folder);
if (!Directory.Exists(OutputDir))
{
Directory.CreateDirectory(OutputDir);
}

//Check if model compile requirements are empty.
if (string.IsNullOrEmpty(this.TimeSetUsed) || string.IsNullOrEmpty(this.ClockPinItem) ||
string.IsNullOrEmpty(this.DataPinItem))
{
Console.WriteLine($"{this.FileName}: Time Set, Clock Pin, and Data Pin cannot be empty .");
return;
}

//this.BuildAllDPatSrcDataForModes(); //Needed before save.
this.SaveDPATSRC(FileName, OutputDir);

//Check to see if the seDPinPatternCompiler_64.exe has been installed or exists in the installation
path.
string compilerDDSToolPath = @"C:\Program Files\IVI
Foundation\IVI\Drivers\seDPin\PatternCompiler\seDPinPatternCompiler_64.exe"; //Universal
Installation Path.
if (!File.Exists(compilerDDSToolPath)) //If exe file does not exist, throw a message.
{
MessageBox.Show($"{compilerDDSToolPath} does not exist or was never installed...");
return;
}
//Check to make sure the .pinmap file for SEDPIN usage exists and then use it as a parameter.
string SeDpinMapFilePath = Path.ChangeExtension(pinMapFilePath, ".pinmap");
if (!File.Exists(SeDpinMapFilePath))
{
MessageBox.Show($"{SeDpinMapFilePath} does not exist, cannot continue without a .pinmap
file");
```

```
return;
}
Compile_To_DPAT(FileName, SeDpinMapFilePath, OutputDir);
}
#region Compile to DPAT
// Define static variables shared by class methods.
private static StringBuilder ShellOutput = null;
private static StringBuilder ShellErrorsToWrite = null;
private static int numOutputLines = 0;
private void Compile_To_DPAT(string FileName, string SeDpinMapFilePath, string
OutputDirectory)
{
try
{
// create and start a Stopwatch instance
Stopwatch stopwatch = Stopwatch.StartNew();

#region Test Code
using (Process myProcess = new Process())
{
myProcess.StartInfo.FileName = @"powershell.exe";
myProcess.StartInfo.UseShellExecute = false; //Setting this to true will not allow us to redirect the
input and output.
myProcess.StartInfo.RedirectStandardInput = true;
myProcess.StartInfo.WorkingDirectory = @"C:\Program Files\IVI
Foundation\IVI\Drivers\seDPin\PatternCompiler"; //compilerDDSToolPath parent directory.
myProcess.StartInfo.RedirectStandardOutput = true;
myProcess.StartInfo.RedirectStandardError = true; //TEST
myProcess.StartInfo.CreateNoWindow = true;
// Set our event handler to asynchronously read the sort output.
myProcess.OutputDataReceived += OutputHandler;
//myProcess.ErrorDataReceived += ErrorHandler;

ShellOutput = new StringBuilder(); //Must be set on each call since var is static.
ShellErrorsToWrite = new StringBuilder(); //Must be set on each call since var is static.

myProcess.Start();
StreamWriter myStreamWriter = myProcess.StandardInput;

// Start the asynchronous read of the sort output stream.
myProcess.BeginOutputReadLine();
```

```
foreach (GenericMode mode in this.ModeCollection)
{
string AlteredModeName = Regex.Replace(mode.Name, "[^a-zA-Z0-9_]+", "_"); //Replace all
special characters in the mode name for compiler reasons "parenthesis".
string dPatSrcFilePath = Path.Combine(OutputDirectory, string.Format("{0}.dpatsrc",
AlteredModeName));

string outputFilePath = Path.Combine(OutputDirectory, string.Format("{0}.dpat",
AlteredModeName)); //.dpat file path

string Command = $@".\seDPinPatternCompiler_64.exe -p '{dPatSrcFilePath}' -m
'{SeDpinMapFilePath}' -o '{outputFilePath}' -d '{OutputDirectory}' -suppress_log";
myStreamWriter.WriteLine(Command);
}
myStreamWriter.Close();

// Wait for the sort process to write the sorted text lines.
myProcess.WaitForExit();
}
#endregion

//Write all shell output to the compile logs.
using (StreamWriter writetext = new StreamWriter(Path.Combine(OutputDirectory,
"AllCompileLogs.txt")))
{
writetext.Write(ShellOutput);
}
stopwatch.Stop();

#region Write final result to console
if (string.IsNullOrEmpty(ShellErrorsToWrite.ToString()))
{
Console.WriteLine($"Successfully compiled all .dpat files (in {stopwatch.ElapsedMilliseconds} ms)
using {FileName}");
}
else
{
Console.WriteLine(ShellErrorsToWrite.ToString());
Console.WriteLine($"Failed to compile all .dpat files using {FileName}");
}
#endregion Write final result to console
}
```

```
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
}
private static void OutputHandler(object sendingProcess,
System.Diagnostics.DataReceivedEventArgs outLine)
{
// Collect the sort command output.
if (!String.IsNullOrEmpty(outLine.Data))
{
numOutputLines++;

#region Error String Check
List<string> errorWords = new List<string>() { "[Failed]", "[failed]", "[Missing]", "[missing]", "[Error]",
"[error]" };
bool lineContainsError = errorWords.Any(c => outLine.Data.Contains(c));
if (lineContainsError)
{
//Console.WriteLine($"PowerShell: {outLine.Data}"); //Console.WriteLine() won't work since
standard input is redirected to the PowerShell process.
ShellErrorsToWrite.Append(Environment.NewLine + outLine.Data);
}
#endregion

// Add the text to the collected output.
ShellOutput.Append(Environment.NewLine + $"[{numOutputLines}] - {outLine.Data}");
}
}
#endregion

#region Translate to DPATSRC
public string CompileModeToDPATSRC(GenericMode mode)
{
StringBuilder sb = new StringBuilder();
try
{
// Build the pattern string
//sb.AppendLine("");
sb.AppendLine("dpat_format_version 1.1;");
sb.AppendLine(string.Format("timeset {0};", this.TimeSetUsed));
sb.AppendLine("svm_only;");
```

```
sb.AppendLine("");
sb.AppendLine("pattern " + mode.Name + " (" + this.ClockPinItem + ":b," + this.DataPinItem +
":b)");
sb.AppendLine("{");
//writetext.WriteLine("start:"); //Causes problem on update for loading sub-patterns or something
on the instrument.

//Append sections of pattern.
foreach (GenericModeVector vector in mode.ModeVectors)
{
int modeVectorIndex = mode.ModeVectors.IndexOf(vector);
ObservableCollection<Operation> MasterOps = vector.VectorizedData;

//NoNeed or Empty bypass
if (MasterOps.Count() <= 0)
{
continue;
}

//Sort sections of the operations w/ LINQ.
List<Operation> startSequence = MasterOps.Where(x => x.Section ==
VectorSection.Start_Sequence_Control).ToList();
List<Operation> commandSequence = MasterOps.Where(x => x.Section ==
VectorSection.Command).ToList();
List<Operation> registerAddressSequence = MasterOps.Where(x => x.Section ==
VectorSection.Register_Address).ToList(); //Separated by parity bits, no comments.
List<Operation> maskBitSequence = MasterOps.Where(x => x.Section ==
VectorSection.Mask_EightBit).ToList();
List<Operation> dataSequence = MasterOps.Where(x => x.Section ==
VectorSection.Data).ToList(); //Separated by parity bits, no comments.
List<Operation> busPark = MasterOps.Where(x => x.Section ==
VectorSection.Bus_Park).ToList();

//Being Writing the DPATSRC.
sb.AppendLine("// Write reg 0x" + mode.ModeVectors[modeVectorIndex].RegisterValue + " data
0x" + mode.ModeVectors[modeVectorIndex].InputValue);
//Print condensed.
if (this.IsCondensed)
{
if(startSequence.Count != 0)
{
sb.AppendLine("// Start sequence control");
```

```
AppendOperationSequence(sb, startSequence);
}
if(commandSequence.Count != 0)
{
sb.AppendLine("// Command");
AppendOperationSequence(sb, commandSequence);
}
if(registerAddressSequence.Count != 0)
{
sb.AppendLine("// Reg Addr");
AppendOperationSequence(sb, registerAddressSequence);
}
if (registerAddressSequence.Count != 0)
{
sb.AppendLine("// Mask");
AppendOperationSequence(sb, maskBitSequence);
}
if (dataSequence.Count != 0)
{
sb.AppendLine("// Data");
AppendOperationSequence(sb, dataSequence);
}
if(busPark.Count != 0)
{
sb.AppendLine("// Bus Park");
AppendOperationSequence(sb, busPark);
}

}
else //Print expanded.
{
if (startSequence.Count != 0)
{
sb.AppendLine("// Start sequence control");
foreach (var sVector in startSequence)
{
sb.AppendLine(sVector.ToString());
}
}
if (commandSequence.Count != 0)
{
sb.AppendLine("// Command");
```

```
foreach (var cVector in commandSequence)
{
sb.AppendLine(cVector.ToString());
}
}
if (registerAddressSequence.Count != 0)
{
sb.AppendLine("// Reg Addr");
foreach (var raVector in registerAddressSequence)
{
sb.AppendLine(raVector.ToString());
}
}
if (maskBitSequence.Count != 0)
{
sb.AppendLine("// Mask");
foreach (var maskVector in maskBitSequence)
{
sb.AppendLine(maskVector.ToString());
}
}
if (dataSequence.Count != 0)
{
sb.AppendLine("// Data");
foreach (var dVector in dataSequence)
{
sb.AppendLine(dVector.ToString());
}
}
if(busPark.Count != 0)
{
sb.AppendLine("// Bus Park");
foreach (var bpVector in busPark)
{
sb.AppendLine(bpVector.ToString());
}
}

}

}
```

```
sb.AppendLine($"HALT\t\t\t\t\t\t  >\t{"-"}\t\tX{"  "}X;"); //Extra HALT line to fix "timing generator"
error (forwarded email from jerry).
sb.AppendLine("}");

return sb.ToString();
}
catch (Exception ex)
{
Console.WriteLine("Vectors failed to compile...");
return sb.ToString();
}
}
public void Compile_Single_Mode_(GenericMode modeToBuild)
{
//Compiles mode's Vectorized data into DPATSRC
modeToBuild.DPatSrcData = CompileModeToDPATSRC(modeToBuild);
}
public void CompileAllModes()
{
foreach (GenericMode mode in this.ModeCollection)
{
//Compiles all Vectorized data into DPATSRC for every mode.
mode.DPatSrcData = CompileModeToDPATSRC(mode);
}
}

//Takes the input values from data vectors and builds/sets the operations needed to compile the
DPATSRC vectors into a pattern text.
#region Vectorize Overload Method Group
public void Vectorize_All_Modes_For_Pattern()
{
foreach (GenericMode mode in this.ModeCollection)
{
Vectorize_All_For_Mode(mode);
}
}
public void Vectorize_All_For_Mode(GenericMode mode)
{
foreach (GenericModeVector vector in mode.ModeVectors)
{
this.Vectorize_Single(mode, vector);
}
```

```
}

public void Vectorize_Single(GenericModeVector modeVector)
{

ObservableCollection<Operation> MasterOps = modeVector.VectorizeInput(this.TimeSetUsed);
modeVector.VectorizedData.Clear();
foreach (Operation op in MasterOps)
{
modeVector.VectorizedData.Add(op);
}
}
public void Vectorize_Single(GenericMode parentMode, GenericModeVector modeVector)
{
int modeVectorIndex = parentMode.ModeVectors.IndexOf(modeVector);
ObservableCollection<Operation> MasterOps = modeVector.VectorizeInput( this.TimeSetUsed);
modeVector.VectorizedData.Clear();
foreach (Operation op in MasterOps)
{
modeVector.VectorizedData.Add(op);
}
}
# endregion Vectorize Overload Method Group

//Saves the DPATSRC text data from every mode to it's own pattern file.
private void SaveDPATSRC(string FileName, string OutputDirectory)
{
try
{
foreach (GenericMode mode in this.ModeCollection)
{
string NewModeName = Regex.Replace(mode.Name, "[^a-zA-Z0-9_]+", "_"); //Replace all special
characters in the mode name for compiler reasons "parenthesis".
string NewFilePath = Path.Combine(OutputDirectory, string.Format("{0}.dpatsrc",
NewModeName));
using (StreamWriter writetext = new StreamWriter(NewFilePath))
{
mode.DPatSrcData = CompileModeToDPATSRC(mode);
writetext.Write(mode.DPatSrcData);
}
}
}
```

```
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }

}

#region Old non dynamic DPATSRC Translation code
///private void SaveDPATSRC(string FileName, string OutputDirectory)
///{
///    try
///    {
///        // create and start a Stopwatch instance
///        Stopwatch stopwatch = Stopwatch.StartNew();
///
///        foreach (Mode mode in this.ModeCollection)
///        {
///            string NewModeName = Regex.Replace(mode.Name, "[^a-zA-Z0-9_]+", "_"); //Replace all special characters in the mode name for compiler reasons "parenthesis".
///            string NewFilePath = Path.Combine(OutputDirectory, string.Format("{0}.dpatsrc", NewModeName));
///            using (StreamWriter writetext = new StreamWriter(NewFilePath))
///            {
///                writetext.WriteLine("");
///                writetext.WriteLine("dpat_format_version 1.1;");
///                writetext.WriteLine(string.Format("timeset {0};", this.TimeSetUsed));
///                writetext.WriteLine("svm_only;");
///                writetext.WriteLine("");
///                writetext.WriteLine(string.Format("pattern {0} ({1}:b,{2}:b)", NewModeName, this.ClockPinItem, this.DataPinItem));
///                writetext.WriteLine("{");
///                //writetext.WriteLine("start:"); //Causes problem on update for loading sub-patterns or something on the instrument.
///                for (int dataIndex = 0; dataIndex < mode.RegData.Count; dataIndex++)
//foreach(string data in mode.RegData)
///                {
///                    string data = mode.RegData[dataIndex];
///                    string register = Regex.Replace(Registers[dataIndex], "_", "", RegexOptions.Compiled); //[^a - zA - Z0 - 9_.]+
///
///                    #region Pre-Process Data Checks
///    Break if the data is not required. Should break if the data string does not contain an x or perhaps a 0x : TBD.
///    if (data == "no need" || data == "No Need" || data == "NN" || data == "nn") { continue; }
///                    else if (string.IsNullOrEmpty(data)) //Break if the data is null or empty then send a
```

```
warning to the user.
///                 {
///                     Console.WriteLine($"WARNING: {Path.GetFileName(FileName)}: Mode
{mode.Name} contains a blank hex value for Register {register}"); continue;
///                 }
///                 //For some reason it allows the letter "a" through and breaks the entire process.
///                 if (!OnlyHexInString(data)) //Checks if the data only contains only C-style hex
notation (0xFF) values
///                 {
///                     Console.WriteLine($"WARNING: {Path.GetFileName(FileName)}: Mode
{mode.Name} contains invalid hex data for Register {register}"); continue;
///                 }
///                 #endregion Pre-Process Data Checks
///
///                 writetext.WriteLine(string.Format("// Write reg {0} data {1}", register, data));
///                 writetext.WriteLine("// Start sequence control");
///                 writetext.WriteLine($"REPEAT(3)                    >  {this.TimeSetUsed}     0   X;");
///                 writetext.WriteLine($"REPEAT(6)                    >  {this.TimeSetUsed}     0   0;");
///                 writetext.WriteLine($"                          >  {this.TimeSetUsed}     0   1;");
///                 writetext.WriteLine($"                          >  {this.TimeSetUsed}     0   0;");
///                 writetext.WriteLine("//Command");
///                 writetext.WriteLine($"REPEAT(4)                    >  {this.TimeSetUsed}     1   1;");
//Slave address
///                 int[] cArray = TranslateToBinary(register, "reg");
///                 for (int i = 0; i < cArray.Length; i++) //reg address C7, C6, C5, C4, C3, C2, C1, C0
///                 {
///                     writetext.WriteLine($"                          >  {this.TimeSetUsed}     1
{cArray[i]};");
///                 }
///                 int cParityBit = CalculateParityBit(cArray);
///                 writetext.WriteLine($"                          >  {this.TimeSetUsed}     1
{cParityBit};");
///                 writetext.WriteLine("//Data");
///                 int[] dArray = TranslateToBinary(data, "data");
///                 for (int i = 0; i < dArray.Length; i++)//data D7, D6, D5, D4, D3, D2, D1, D0
///                 {
///                     writetext.WriteLine($"                          >  {this.TimeSetUsed}     1
{dArray[i]};");
///                 }
///                 int dParityBit = CalculateParityBit(dArray);
///                 writetext.WriteLine($"                          >  {this.TimeSetUsed}     1
{dParityBit};"); //Write in the parity bit.
```

```
///                 writetext.WriteLine("//Bus Park");
///                 writetext.WriteLine($"                          >  {this.TimeSetUsed}     1   0;");
///                 writetext.WriteLine($"                          >  {this.TimeSetUsed}     0   X;");
///                 writetext.WriteLine($"                          >  {this.TimeSetUsed}     X   X;");
///             }
///             writetext.WriteLine($"HALT                          >  {"-"}         X   X;"); //Extra HALT line
to fix "timing generator" error (forwarded email from jerry).
///             writetext.WriteLine("}");
///         }
///     }
///     stopwatch.Stop();
///     Console.WriteLine($"Successfully compiled all modes to .dpatsrc files (in
{stopwatch.ElapsedMilliseconds} ms) using {this.FilePath}");
///  }
///  catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
///}

////Counts the total number of ones in the 8 bit array given to determine and return the parity bit.
//public static int CalculateParityBit(int[] bitArray)
//{
//  int totalOnes = 0;
//  for (int i = 0; i < bitArray.Length; i++)
//  {
//      totalOnes += bitArray[i] == 1 ? 1 : 0;
//  }
//  int resultVal = totalOnes % 2 == 0 ? 1 : 0;  //odd number returns 0 as parity bit while even
numbers return 1 as parity bit
//  return resultVal;
//}
//
////Translates the register codes and the data codes to return the data in the order it should be
serialized.
//public static int[] TranslateToBinary(string asciiCode, string regORdata)
//{
//  int[] returnValue;
//
//  char[] code = asciiCode.ToCharArray();
//  switch (regORdata)
//  {
//      case "reg":
//          returnValue = new int[8];
//          returnValue[0] = 0; //command C7  (010: Write, 011: Read)
```

```
//            returnValue[1] = 1; //command C6
//            returnValue[2] = 0; //command C5
//            char[] cBinChars = hex2binary((code[2].ToString() + code[3].ToString())).ToCharArray();
//            for (int i = 3; i < cBinChars.Length; i++)
//            {
//                returnValue[i] = Int32.Parse(cBinChars[i].ToString()) ;
//            }
//            break;
//        case "data":
//            returnValue = new int[8];
//            char[] dBinChars = hex2binary((code[2].ToString() + code[3].ToString())).ToCharArray();
//            for (int i = 0; i < dBinChars.Length; i++)
//            {
//                returnValue[i] = Int32.Parse(dBinChars[i].ToString());
//            }
//            break;
//        default: returnValue = null; break;
//    }
//
//    return returnValue;
//}
////Helper method converting hex to binary.
//public static string hex2binary(string hexvalue)
//{
//    String result = hexvalue
//    .Aggregate(new StringBuilder(), (builder, c) =>
//    builder.Append(Convert.ToString(Convert.ToInt32(c.ToString(), 16), 2).PadLeft(4, '0')))
//    .ToString();
//
//    return result;
//}

//Helper method.
#endregion Old non dynamic DPATSRC Translation code
public static bool OnlyHexInString(string data)
{
// For C-style hex notation (0xFF) you can use @"\A\b(0[xX])?[0-9a-fA-F]+\b\Z"
return Regex.IsMatch(data, @"\A\b(0[xX])?[0-9a-fA-F]+\b\Z");
}

// Appends a sequence of operations to the StringBuilder, using REPEAT opcodes where possible
public void AppendOperationSequence(StringBuilder sb, List<Operation> sequence)
```

```
{
try
{
if (sequence != null && sequence.Count() > 0)
{
int repeatCount = 1;
for (int i = 1; i < sequence.Count; i++)
{
if (sequence[i].RepeatEquals(sequence[i - 1]))
{
repeatCount++;
}
else
{
if (repeatCount > 1)
{
sequence[i - 1].Opcode = "REPEAT(" + repeatCount + ")";
sb.AppendLine(sequence[i - 1].ToString());
sequence[i - 1].Opcode = "";
}
else
{
sb.AppendLine(sequence[i - 1].ToString());
}
repeatCount = 1;
}
}
if (repeatCount > 1)
{
sequence[sequence.Count - 1].Opcode = "REPEAT(" + repeatCount + ")";
sb.AppendLine(sequence[sequence.Count - 1].ToString());
sequence[sequence.Count - 1].Opcode = "";
}
else
{
sb.AppendLine(sequence[sequence.Count - 1].ToString());
}
}
}
catch (Exception ex)
{
Console.WriteLine($"Repeats failed to condense: {ex.Message}");
```

```
}
}
#endregion

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
OnChangeOccured();
}
#endregion

public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}

}

[Serializable]
public enum VectorSection
{
Start_Sequence_Control,
Command,

Register_Address,
Mask_EightBit,

Data,
Bus_Park,
}

// Represents an operation in the pattern
[Serializable]
public class Operation
{
private string _opcode = "";

public int VectorNumber { get; set; }
public string Opcode
```

```
{
get { return _opcode; }
set
{
_opcode = value != null ? value : "";
}
}

public string TimeSet { get; set; }
public string Clock { get; set; }
public string Data { get; set; }

public string Comment { get; set; }
public VectorSection Section { get; set; }

//GUI PROPERTY
[XmlIgnore]
public string Original_Input { get; set; }

#region Constructors
public Operation() { } //Deserialization Constructor.
public Operation(string clock, string data)
{
Clock = clock;
Data = data;
}
public Operation(string name, string clock, string data)
{
TimeSet = name;
Clock = clock;
Data = data;
}
#endregion

public override string ToString()
{
int space = 32 - Opcode.Length;

return Opcode + ">\t".PadLeft(space) + TimeSet + "\t" + Clock + "  " + Data + ";" + "   " + Comment;
}

public bool RepeatEquals(Operation obj)
```

```
{
Operation other = obj as Operation;
if (other == null)
{
return false;
}
return Opcode == other.Opcode && TimeSet == other.TimeSet && Clock == other.Clock && Data
== other.Data;
}
}
}
```

MappingModel.cs

```
using MerlinTestStudio.DataModels;
using Newtonsoft.Json;
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data
{
[Serializable]
public class MappingModel : INotifyPropertyChanged //Change to "PinMapping" to not get
confused with Parameter Mapping.
{
#region Private memebers
private string _selectedIO;
private string _name;
private Instrument _instrument;
#endregion

#region Events
public event Action<MappingModel> OnChangeOccured;
#endregion

#region Public members
[XmlIgnore]
public Instrument Instrument
{
get { return _instrument; }
set
```

```
{
_instrument = value;
OnPropertyChanged("Instrument");
if(value != null)
{
InstrumentName = value.InstrumentName;
}
}
}
public string InstrumentName
{
get { return _name; }
set { _name = value; OnPropertyChanged("InstrumentName"); } //Call
OnPropertyChanged("Ports"); ??
}
public string IO
{
get { return _selectedIO; }
set { _selectedIO = value; OnPropertyChanged("IO"); OnChangeOccured?.Invoke(this); }
}

[JsonIgnore]
[XmlIgnore] //Temporary until a better port selection method is accomplished.
public ObservableCollection<string> Ports
{
get { return BackendConstants.GetPorts(Instrument != null ? Instrument.InstrumentName :
"BlankDefaultPorts"); }
}
public int Board { get; set; }
#endregion

#region Static Mapping Model Methods
public static MappingModel GetMappingModelFromMapping(Mapping fromMapping)
{
MappingModel modelItem = new MappingModel();

MappingModel.DataSwap(fromMapping, modelItem);

return modelItem;
}
public static Mapping GetMappingFromMappingModel(MappingModel fromModel)
{
```

```
Mapping Item = new Mapping();

MappingModel.DataSwap(fromModel, Item);

return Item;
}


/// <summary>
/// Swaps the current offsets data from the Test object to the OffsetLimit object.
/// </summary>
/// <param name="fromTest"></param>
/// <param name="toLimit"></param>
public static void DataSwap(MappingModel fromModel, Mapping toMapping)
{
toMapping.InstrumentName = fromModel.InstrumentName;
toMapping.IO = fromModel.IO;
toMapping.Board = fromModel.Board;
}

/// <summary>
/// Swaps the current offsets data from the OffsetLimit object to the Test object.
/// </summary>
/// <param name="fromLimit"></param>
/// <param name="toTest"></param>
public static void DataSwap(Mapping fromMapping, MappingModel toModel)
{
toModel.InstrumentName = fromMapping.InstrumentName;
toModel.IO = fromMapping.IO;
toModel.Board = fromMapping.Board;
}
#endregion

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
```

}
}

PinMapModel.cs

```
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models.PinModels;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Collections.Specialized;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Xml;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data.Models
{
public enum PinMapSection
{
Rf_Pins,
Digital_Pins,
Power_Supply_Pins,

NoSection
}


#region Pin Map Undo/Redo Commands
public class RemoveSiteCmd : IDoCommand
{
private event Action RemoveSite;
private event Action ReverseAction;
private Site SiteToRemove;
private PinMapModel DataModel;
private List<MappingModel> rRfMappings;
private List<MappingModel> rDigMappings;
```

```
private List<MappingModel> rPwrMappings;
public RemoveSiteCmd(PinMapModel dataModel, Action removeAction, Action reverseAction, Site
siteToRemove)
{
RemoveSite = removeAction;
ReverseAction = reverseAction;
SiteToRemove = siteToRemove;
DataModel = dataModel;
}

public void Execute()
{
//Removes the site mappings data.
int SiteID = SiteToRemove.SiteID - 1; //Minus 1 to use as mappings index.
var rfMappingsToRemove = new List<MappingModel>();
foreach (Data.PinModel p in DataModel.RfPins)
{
rfMappingsToRemove.Add(p.mappings[SiteID]);
p.mappings.RemoveAt(SiteID);
}
rRfMappings = rfMappingsToRemove;

var digMappingsToRemove = new List<MappingModel>();
foreach (Data.PinModel p in DataModel.DigitalPins)
{
digMappingsToRemove.Add(p.mappings[SiteID]);
p.mappings.RemoveAt(SiteID);
}
rDigMappings = digMappingsToRemove;

var pwrMappingsToRemove = new List<MappingModel>();
foreach (Data.PinModel p in DataModel.PowerSupplyPins)
{
pwrMappingsToRemove.Add(p.mappings[SiteID]);
p.mappings.RemoveAt(SiteID);
}
rPwrMappings = pwrMappingsToRemove;

//Rebinds the GUI sties to the new pin map data.
RemoveSite?.Invoke();
}
```

```
public void Undo()
{
int SiteID = SiteToRemove.SiteID - 1; //Minus 1 to use as mappings index.
for (int i = 0; i < rRfMappings.Count; i++)
{
DataModel.RfPins[i].mappings.Insert(SiteID, rRfMappings[i]);
}
for (int i = 0; i < rDigMappings.Count; i++)
{
DataModel.DigitalPins[i].mappings.Insert(SiteID, rDigMappings[i]);
}
for (int i = 0; i < rPwrMappings.Count; i++)
{
DataModel.PowerSupplyPins[i].mappings.Insert(SiteID, rPwrMappings[i]);
}

ReverseAction?.Invoke();
}
}
public class AddSiteCmd : IDoCommand
{
private event Action AddSite;
private event Action ReverseAction;
private int MaxMappingsCount;
private PinMapModel DataModel;
public AddSiteCmd(PinMapModel dataModel, Action addAction, Action reverseAction, int
maxMappingsCount)
{
AddSite = addAction;
ReverseAction = reverseAction;
DataModel = dataModel;
MaxMappingsCount = maxMappingsCount;
}

public void Execute()
{
//Add the mappings to each pin in every section with a default instrument name.
///MUST select default instrument based on the first site mapping instrument.
foreach (Data.PinModel P in DataModel.RfPins)
{
Instrument inst = P.mappings.FirstOrDefault().Instrument;
P.mappings.Add(new Data.MappingModel() { Instrument = inst });
```

```
}
foreach (Data.PinModel P in DataModel.DigitalPins)
{
Instrument inst = P.mappings.FirstOrDefault().Instrument;
P.mappings.Add(new Data.MappingModel() { Instrument = inst });
}
foreach (Data.PinModel P in DataModel.PowerSupplyPins)
{
Instrument inst = P.mappings.FirstOrDefault().Instrument;
P.mappings.Add(new Data.MappingModel() { Instrument = inst });
}

//Rebinds the GUI sties to the new pin map data.
AddSite?.Invoke();
}

public void Undo()
{
//Removes the site mappings data.
int SiteID = MaxMappingsCount - 1; //Minus 1 to use as mappings index.
foreach (Data.PinModel p in DataModel.RfPins)
{
p.mappings.Remove(p.mappings.Last());
}
foreach (Data.PinModel p in DataModel.DigitalPins)
{
p.mappings.Remove(p.mappings.Last());
}
foreach (Data.PinModel p in DataModel.PowerSupplyPins)
{
p.mappings.Remove(p.mappings.Last());
}

ReverseAction?.Invoke();
}
}
public class AddPinCmd : IDoCommand
{
private string NameOrNull;
private PinMapSection Section;
private PinMapModel DataModel;
private Data.PinModel newPin;
```

```
public AddPinCmd(PinMapModel dataModel, PinMapSection section, string nameOrNull)
{
NameOrNull = nameOrNull;
Section = section;
DataModel = dataModel;
}

public void Execute()
{
Data.PinModel P = new Data.PinModel() { PinName = "Default" };

if (string.IsNullOrEmpty(NameOrNull)) // If null use a uniquely generated pin name.
{
//Unique name setting.
switch (Section)
{
case PinMapSection.Rf_Pins:
P.PinName = PinMapModel.GetUniquePinName(DataModel.RfPins, "Rf Pin");
break;
case PinMapSection.Digital_Pins:
P.PinName = PinMapModel.GetUniquePinName(DataModel.DigitalPins, "Dig Pin");
break;
case PinMapSection.Power_Supply_Pins:
P.PinName = PinMapModel.GetUniquePinName(DataModel.PowerSupplyPins, "Pwr Pin");
break;
}
}
else { P.PinName = NameOrNull; } //else use the name that was passed in.

//Mapping generation. (Can use some shortening of code...)
switch (Section)
{
case PinMapSection.Rf_Pins:
Instrument RfInst = DataStorage.CurrentSystem.RF_Instruments.FirstOrDefault();
if (DataModel.MaxMappingsCount == 0)
{
P.mappings.Add(new Data.MappingModel() { Instrument = RfInst });
DataModel.RfPins.Add(P);
}
else
{
for (int i = 1; i <= DataModel.MaxMappingsCount; i++)
```

```
{
P.mappings.Add(new Data.MappingModel() { Instrument = RfInst });
}
DataModel.RfPins.Add(P);
}
break;
case PinMapSection.Digital_Pins:
Instrument DigInst = DataStorage.CurrentSystem.PXI_Instruments.FirstOrDefault();
if (DataModel.MaxMappingsCount == 0)
{
P.mappings.Add(new Data.MappingModel() { Instrument = DigInst });
DataModel.DigitalPins.Add(P);
}
else
{
for (int i = 1; i <= DataModel.MaxMappingsCount; i++)
{
P.mappings.Add(new Data.MappingModel() { Instrument = DigInst });
}
DataModel.DigitalPins.Add(P);
}
break;
case PinMapSection.Power_Supply_Pins:
Instrument PowInst = DataStorage.CurrentSystem.Power_Instruments.FirstOrDefault();
if (DataModel.MaxMappingsCount == 0)
{
P.mappings.Add(new Data.MappingModel() { Instrument = PowInst});
DataModel.PowerSupplyPins.Add(P);
}
else
{
for (int i = 1; i <= DataModel.MaxMappingsCount; i++)
{
P.mappings.Add(new Data.MappingModel() { Instrument = PowInst });
}
DataModel.PowerSupplyPins.Add(P);
}
break;
}

newPin = P;
}
```

```
public void Undo()
{
switch (Section)
{
case PinMapSection.Rf_Pins:
DataModel.RfPins.Remove(newPin);
break;
case PinMapSection.Digital_Pins:
DataModel.DigitalPins.Remove(newPin);
break;
case PinMapSection.Power_Supply_Pins:
DataModel.PowerSupplyPins.Remove(newPin);
break;
}
}
}
public class RemovePinCmd : IDoCommand
{
private PinMapSection Section;
private PinMapModel DataModel;
private Data.PinModel PinToRemove;
public RemovePinCmd(PinMapModel dataModel, PinMapSection section, Data.PinModel
pinToRemove)
{
PinToRemove = pinToRemove;
Section = section;
DataModel = dataModel;
}

public void Execute()
{
switch (Section)
{
case PinMapSection.Rf_Pins:
DataModel.RfPins.Remove(PinToRemove);
break;
case PinMapSection.Digital_Pins:
DataModel.DigitalPins.Remove(PinToRemove);
break;
case PinMapSection.Power_Supply_Pins:
DataModel.PowerSupplyPins.Remove(PinToRemove);
```

```
break;
}
}

public void Undo()
{
switch (Section)
{
case PinMapSection.Rf_Pins:
DataModel.RfPins.Add(PinToRemove);
break;
case PinMapSection.Digital_Pins:
DataModel.DigitalPins.Add(PinToRemove);
break;
case PinMapSection.Power_Supply_Pins:
DataModel.PowerSupplyPins.Add(PinToRemove);
break;
}
}
}
#endregion

/// <summary>
/// GUI Model.
/// </summary>
public class PinMapModel : INotifyPropertyChanged, IFileData
{
public CommandHandler CmdHandler = new CommandHandler();

#region Private members
private string _filePath;
private ObservableCollection<Data.PinModel> _rfPins = new ObservableCollection<PinModel>();
private ObservableCollection<Data.PinModel> _digitalPins = new
ObservableCollection<PinModel>();
private ObservableCollection<Data.PinModel> _powerSupplyPins = new
ObservableCollection<PinModel>();
#endregion
public string FileName { get { return Path.GetFileNameWithoutExtension(this.FilePath); } }
public string FilePath { get { return _filePath; } set { _filePath = value;
OnPropertyChanged("FilePath"); OnPropertyChanged("FileName"); } }
public double Version { get; set; }
//public int Sites
```

```
public ObservableCollection<Data.PinModel> RfPins
{
get { return _rfPins; } set { _rfPins = value; OnPropertyChanged("RfPins"); }
}
public ObservableCollection<Data.PinModel> DigitalPins
{
get { return _digitalPins; } set { _digitalPins = value; OnPropertyChanged("DigitalPins"); }
}
public ObservableCollection<Data.PinModel> PowerSupplyPins
{
get { return _powerSupplyPins; } set { _powerSupplyPins = value;
OnPropertyChanged("PowerSupplyPins"); }
}

[XmlIgnore]
public int MaxMappingsCount
{
get
{
if (this.RfPins.Count == 0)
{
if (this.DigitalPins.Count() == 0)
{
if (this.PowerSupplyPins.Count() == 0)
{
return 0;
}
else { return this.PowerSupplyPins.Max((x) => x.mappings.Count); }
}
else { return this.DigitalPins.Max((x) => x.mappings.Count); }
}
else
{ return this.RfPins.Max((x) => x.mappings.Count); }
}
}

#region Events
public  event Action SiteAdded;
public event Action SiteRemoved;

public event Action ChangeOccured;
```

```
public event Action CommitEditBeforeSave;
#endregion

#region Constructor
public PinMapModel()
{
this.Version = 1;
RfPins.CollectionChanged += Pin_CollectionChanged;
DigitalPins.CollectionChanged += Pin_CollectionChanged;
PowerSupplyPins.CollectionChanged += Pin_CollectionChanged;
}
#endregion

private void Pin_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
{
if (e.NewItems != null)
{
foreach (PinModel item in e.NewItems)
{
item.PropertyChanged += this.MyType_PropertyChanged;
}
OnChangeOccured();
}
if (e.OldItems != null)
{
foreach (PinModel item in e.OldItems)
{
item.PropertyChanged -= this.MyType_PropertyChanged;
}
OnChangeOccured();
}
}
private void MyType_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
OnChangeOccured();
}
#region Public Methods
public void ValidatePinMapData(Enum[] mapSectionsToCheck, out bool isValid)
{
foreach (PinMapSection section in mapSectionsToCheck)
{
ObservableCollection<Data.PinModel> actingSection = new ObservableCollection<PinModel>();
```

```
switch (section)
{
case PinMapSection.Rf_Pins:  actingSection = this.RfPins; break;
case PinMapSection.Digital_Pins: actingSection = this.DigitalPins; break;
case PinMapSection.Power_Supply_Pins: actingSection = this.PowerSupplyPins; break;
}

#region Temp Clear of all errors in order to re-check
foreach(var pin in actingSection)
{
pin.Status = ValidationStatus.Valid;
pin.ToolTipString = "";
}
#endregion

#region Unique Pin Name and Pin Alias Check
Dictionary<Func<IEnumerable<PinModel>, bool>, string> rules = new
Dictionary<Func<IEnumerable<PinModel>, bool>, string>()
{
{
s => {
List<string> duplicatePinNames = s.Where(x => string.IsNullOrEmpty(x.PinAlias)).GroupBy(x =>
x.PinName).Where(g => g.Count() > 1).Select(g => g.Key).ToList();
if (duplicatePinNames.Count() > 0)
{
foreach (var pin in s.Where(x => string.IsNullOrEmpty(x.PinAlias)))
{
if (duplicatePinNames.Contains(pin.PinName))
{
pin.Status = ValidationStatus.Invalid;
pin.ToolTipString = "Pins, without a Pin Alias, cannot have the same Pin Name.";
}
}
return false;
}
return true;
},
"PinMap: Pins, without a Pin Alias, cannot have the same Pin Name."
},
{
s => {
List<string> duplicatePinAliases =s.Where(x => !string.IsNullOrEmpty(x.PinAlias)).GroupBy(x =>
```

```
x.PinAlias).Where(g => g.Count() > 1).Select(g => g.Key).ToList();
if (duplicatePinAliases.Count() > 0)
{
foreach (var pin in s.Where(x => !string.IsNullOrEmpty(x.PinAlias)))
{
if (duplicatePinAliases.Contains(pin.PinAlias))
{
pin.Status = ValidationStatus.Invalid;
pin.ToolTipString = "Pins, with a Pin Alias, cannot have the same Pin Alias.";
}
}
return false;
}
return true;
},
"PinMap: Pins, with a Pin Alias, cannot have the same Pin Alias."
},
{
s => {
var PinsNamesWithNoAliases = s.Where(x => string.IsNullOrEmpty(x.PinAlias)).GroupBy(x =>
x.PinName).Select(g => g.Key);
var PinNamesWithAliases = s.Where(x => !string.IsNullOrEmpty(x.PinAlias)).GroupBy(x =>
x.PinName).Select(g => g.Key);
var commonNames = PinsNamesWithNoAliases.Intersect(PinNamesWithAliases);
if (commonNames.Count() > 0)
{
foreach (var pin in s)
{
if (commonNames.Contains(pin.PinName))
{
pin.Status = ValidationStatus.Invalid;
pin.ToolTipString = "Pins cannot have a Pin Name that matches an Alias-Grouped Pin Name";
}
}
return false;
}
return true;
},
"PinMap: Pins cannot have a Pin Name that matches an Alias-Grouped Pin Name"
}
};
```

```
List<string> errors = Validator.Validate(actingSection, rules);

if (errors.Count() > 0)
{
foreach (var e in errors)
{
//MessageBox.Show(e);
Console.WriteLine(e);
}
isValid = false;
}



//if (duplicatePinNames.Count() > 0)
//{
//    string samePinNames = string.Empty;
//    foreach (string pinName in duplicatePinNames)
//    {
//        samePinNames += string.Format("\n{0}", pinName);
//    }
//    MessageBox.Show(string.Format("PinMap: The following pins, without aliases, cannot have
the same Pin Name or Alias Group Pin Name: {0}", samePinNames));
//    return false;
//}
#endregion

#region Multiple Ports in Site Check
if (actingSection.Any())
{
var errorMsg = "The following sites have many of the same port(s): ";
for (int site = 0; site < actingSection.Max(x => x.mappings.Count); site++)
{
var SiteMappings = actingSection.SelectMany(x => x.mappings.Where(m => m != null &&
x.mappings.IndexOf(m) == site));
var duplicateIOs = SiteMappings.GroupBy(x => x.IO).Where(g => g.Count() > 1).Select(g =>
g.Key).ToList();

if (duplicateIOs.Any())
{
errorMsg += $"\n - Site {site + 1}: {string.Join("  ", duplicateIOs)}";
}
```

```
foreach (var pin in actingSection)
{
if (duplicateIOs.Contains(pin.mappings[site]?.IO))
{
pin.Status = ValidationStatus.Invalid;
pin.ToolTipString = errorMsg;
}
}
}
if (errorMsg != "The following sites have many of the same port(s): ")
{
Console.WriteLine(errorMsg);
isValid = false;
}
}
#endregion

#region Full Site Mappings Check
///Checks to see if all the available sites are mapped and not empty.
bool? allSitesFull = true; //Default true, until a false is hit in the loop below
foreach (Data.PinModel p in actingSection)
{
foreach (Data.MappingModel m in p.mappings)
{
if (string.IsNullOrEmpty(m.InstrumentName) || string.IsNullOrEmpty(m.IO))
{
p.Status = ValidationStatus.Invalid;
p.ToolTipString = "Unmapped sites detected, please remove empty sites or fill in the blanks.";

allSitesFull = false;
}
}
}
if (allSitesFull == false)
{
Console.WriteLine("PinMap: Unmapped sites detected, please remove empty sites or fill in the
blanks.");
isValid = false;
}
#endregion

}
```

```
isValid = true; //All checks clear, return valid.
}

//Helper method.
public static PinMapSection GetPinMapSectionEnum(string CoString)
{
switch (CoString)
{
case "RF Pins":
case "RfPins":
case "Rf_Pins":
return PinMapSection.Rf_Pins;
case "Digital Pins":
case "DigitalPins":
case "Digital_Pins":
return PinMapSection.Digital_Pins;
case "Power Supply Pins":
case "PowerSupplyPins":
case "Power_Supply_Pins":
return PinMapSection.Power_Supply_Pins;
default:
return PinMapSection.NoSection; //Shouldn't be used in any case. Implentation is a work around.
}
}
//Helper method.
public static string GetUniquePinName(ObservableCollection<Data.PinModel> pinCollection,
string pinName)
{
int iteration = 1;
bool uniqueName;
string newUniqueName = string.Format("{0} (1)", pinName);
do
{
uniqueName = true;

foreach (Data.PinModel pin in pinCollection)
{
if (pin.PinName == newUniqueName)
{
uniqueName = false;
iteration++;
```

```
newUniqueName = string.Format("{0} ({1})", pinName, iteration);
break;
}
}
} while (uniqueName != true);

return newUniqueName;
}


public void AddPin(PinMapSection pinMapSection, string pinNameOrNull)
{
CmdHandler.AddCommand(new AddPinCmd(this, pinMapSection, pinNameOrNull));
}

public void RemovePin(PinMapSection pinMapSection, PinModel pinToRemove) //Should aslo
add a function that removes based on pin name.
{
CmdHandler.AddCommand(new RemovePinCmd(this, pinMapSection, pinToRemove));
}

public void AddSite()
{
CmdHandler.AddCommand(new AddSiteCmd(this, SiteRemoved, SiteAdded,
MaxMappingsCount));
}

public void RemoveSite(Site siteToRemove)
{
CmdHandler.AddCommand(new RemoveSiteCmd(this, SiteRemoved, SiteAdded,
siteToRemove));
}
#endregion


//May not be needed.
public string SeparateBoardNumberFromInstrumentName(string instrumentName)
{
if (!string.IsNullOrEmpty(instrumentName)) //If it is not null or empty string, then proceed with
regex processing.
{
return instrumentName.Split(' ')[0];
```

```
}
else { return string.Empty; }
}

#region Save/Import
public void SaveToXml(string FilePath)
{
PinMapData pmd = new PinMapData()
{
FilePath = this.FilePath, //May casue confusion with method paramter.
Version = this.Version,
RfPins = this.RfPins.ToList().ConvertAll(new Converter<PinModel,
Pin>(PinModel.GetPinFromPinModel)),
DigitalPins = this.DigitalPins.ToList().ConvertAll(new Converter<PinModel,
Pin>(PinModel.GetPinFromPinModel)),
PowerSupplyPins = this.PowerSupplyPins.ToList().ConvertAll(new Converter<PinModel,
Pin>(PinModel.GetPinFromPinModel))
};
pmd.SaveToXml(FilePath);

#region SDEPIN File
try
{
List<string> InstrumentValues = new List<string>();
foreach (PinModel pin in this.DigitalPins)
{
InstrumentValues.AddRange((pin.mappings.Select(m => m.InstrumentName.Split(' ')[0])).ToList());
}

if (InstrumentValues.Contains("SEDPin32"))
{
string DigFilePath = Path.ChangeExtension(FilePath,
BackendConstants.SEDPin32_PinMap_Extension);
using (XmlTextWriter xmlWriter = new XmlTextWriter(DigFilePath, System.Text.Encoding.UTF8) {
Formatting = Formatting.Indented })
{
xmlWriter.WriteStartDocument();
xmlWriter.WriteStartElement("PinMap");
xmlWriter.WriteAttributeString("xmlns",
"http://www.ni.com/TestStand/SemiconductorModule/PinMap.xsd");
xmlWriter.WriteAttributeString("xmlns:xsi", "http://www.w3.org/2001/XMLSchema-instance");
xmlWriter.WriteAttributeString("schemaVersion", "1.5");
```

```
xmlWriter.WriteStartElement("Instruments");
xmlWriter.WriteStartElement("NIDigitalPatternInstrument");
xmlWriter.WriteAttributeString("name", "DPIN");
xmlWriter.WriteAttributeString("numberOfChannels", "32");
xmlWriter.WriteAttributeString("group", "Digital");
xmlWriter.WriteEndElement();
//foreach (Instrument inst in DataStorage.CurrentSystem.PXI_Instruments)
//{
//    xmlWriter.WriteStartElement("DigitalCard");
//    xmlWriter.WriteAttributeString("name", inst.InstrumentName);
//    xmlWriter.WriteEndElement();
//}
xmlWriter.WriteEndElement();

xmlWriter.WriteStartElement("Pins");
foreach (PinModel pin in this.DigitalPins)
{
if (pin.GetIsSEDPin())
{
string elementName = string.Empty;
List<string> mappingValues = (pin.mappings.Select(m => m.IO)).ToList();
if (mappingValues.Distinct().Count() == 1) //Has only 1 distinct value is system pin.
{
elementName = "SystemPin";
}
else //Has multiple distinct values is DUT pin
{
elementName = "DUTPin";
}
xmlWriter.WriteStartElement(elementName);
xmlWriter.WriteAttributeString("name", pin.PinName);
xmlWriter.WriteEndElement();
}
}
xmlWriter.WriteEndElement();

xmlWriter.WriteStartElement("PinGroups");
xmlWriter.WriteEndElement();

xmlWriter.WriteStartElement("Sites");
for (int i = 0; i < MaxMappingsCount; i++)
```

```
{
xmlWriter.WriteStartElement("Site");
xmlWriter.WriteAttributeString("siteNumber", i.ToString());
xmlWriter.WriteEndElement();
}
xmlWriter.WriteEndElement();

xmlWriter.WriteStartElement("Connections");
foreach (PinModel pin in this.DigitalPins)
{
if (pin.GetIsSEDPin())
{
List<string> mappingValues = (pin.mappings.Select(m => m.IO)).ToList();
if (mappingValues.Distinct().Count() == 1) //Has only 1 distinct value is system Connection.
{
MappingModel firstMap = pin.mappings[0];
int channel = int.Parse(firstMap.IO) - 1;
xmlWriter.WriteStartElement("SystemConnection");
xmlWriter.WriteAttributeString("pin", pin.PinName);
xmlWriter.WriteAttributeString("instrument", "DPIN"); //firstMap.InstrumentName
xmlWriter.WriteAttributeString("channel", channel.ToString());
xmlWriter.WriteEndElement();
}
else //Has multiple distinct values is DUT Connection
{
foreach (MappingModel m in pin.mappings)
{
int channel = int.Parse(m.IO) - 1;
xmlWriter.WriteStartElement("Connection");
xmlWriter.WriteAttributeString("pin", pin.PinName);
xmlWriter.WriteAttributeString("siteNumber", pin.mappings.IndexOf(m).ToString());
xmlWriter.WriteAttributeString("instrument", "DPIN"); //firstMap.InstrumentName
xmlWriter.WriteAttributeString("channel", channel.ToString());
xmlWriter.WriteEndElement();
}
}
}
}
xmlWriter.WriteEndElement();

xmlWriter.WriteEndElement();
xmlWriter.WriteEndDocument();
```

```
}
}
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
#endregion
}
public static PinMapModel ImportFromXml(string FilePath)
{
PinMapData pmd = PinMapData.ImportPinMapXML(FilePath);
PinMapModel rPinMap = new PinMapModel()
{
FilePath = FilePath,
Version = pmd.Version,
RfPins = new ObservableCollection<PinModel>(pmd.RfPins.ConvertAll(new Converter<Pin,
PinModel>(PinModel.GetPinModelFromPin))),
DigitalPins = new ObservableCollection<PinModel>(pmd.DigitalPins.ConvertAll(new
Converter<Pin, PinModel>(PinModel.GetPinModelFromPin))),
PowerSupplyPins = new ObservableCollection<PinModel>(pmd.PowerSupplyPins.ConvertAll(new
Converter<Pin, PinModel>(PinModel.GetPinModelFromPin)))
};
return rPinMap;
}
#endregion

//IFileData
public void Save()
{
CommitEditBeforeSave?.Invoke();
//this.ValidatePinMapData((Enum[])Enum.GetValues(typeof(PinMapSection))); //Temp. If checks
needed.
this.SaveToXml(this.FilePath);
Console.WriteLine(string.Format("Saved: {0}", this.FilePath));
}
public void Load()
{

}
public bool Undo()
```

```
{
CmdHandler.UndoCommand();
//Console.WriteLine("Undo operation not implemented for Pin Map");
//return false;
return true; //if all actions in the command stack have been undone then this should return false.
}
public bool Redo()
{
CmdHandler.RedoCommand();
//Console.WriteLine("Redo operation not implemented for Pin Map");
//return false;
return true;
}

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
OnChangeOccured();
}

#endregion

public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
}

}
```

PinModel.cs

```
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
```

```
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik.Data
{
public enum ValidationStatus
{
Valid = 0, //Default Color
Warning = 1, //Yellow
Invalid = 2 //Red
}



[Serializable]
public class PinModel : INotifyPropertyChanged
{
#region Private members
private string _pinName;
private string _pinAlias;
private ObservableCollection<MappingModel> _mappings = new
ObservableCollection<MappingModel>();
private ValidationStatus _Status = ValidationStatus.Valid;
private string _toolTipString = string.Empty;
#endregion

#region Events

#endregion

#region Public members
public string PinName
{
get { return _pinName; }
set { _pinName = value; OnPropertyChanged("PinName");
OnPropertyChanged("GetPinLabelToUse"); }
}

public string PinAlias
{
get { return _pinAlias; }
set { _pinAlias = value; OnPropertyChanged("PinAlias");
OnPropertyChanged("GetPinLabelToUse"); }
```

```
}

[JsonIgnore]
[XmlIgnore]
public string GetPinLabelToUse //For available pin options binding.
{
get { return string.IsNullOrEmpty(this.PinAlias) ? this.PinName : this.PinAlias; }
}

public ObservableCollection<MappingModel> mappings
{
get { return _mappings; }
set
{
_mappings = value;
OnPropertyChanged("mappings");
//if(value != null)
//{
//    mappings.CollectionChanged += Mapping_CollectionChanged;
//}
}
}
[JsonIgnore]
[XmlIgnore]
public ValidationStatus Status
{
get { return _Status; }
set { _Status = value; OnPropertyChanged("Status"); }
}
[JsonIgnore]
[XmlIgnore]
public string ToolTipString
{
get { return _toolTipString; }
set { _toolTipString = value; OnPropertyChanged("ToolTipString"); }
}

#endregion

#region Constructor
public event Action<MappingModel> MappingChanged;
```

```
public PinModel()
{
mappings.CollectionChanged += Mapping_CollectionChanged; //Not needed, a new enumerable
is set for mappings on load.
}

//~PinModel()
//{
//    //mappings.CollectionChanged -= Mapping_CollectionChanged;
//}
#endregion

private void Mapping_CollectionChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
{
if (e.NewItems != null)
{
foreach (MappingModel item in e.NewItems)
{
item.OnChangeOccured += OnMappingChanged;
}
}

if (e.OldItems != null)
{
foreach (MappingModel item in e.OldItems)
{
item.OnChangeOccured -= OnMappingChanged;
}
}

}

private void OnMappingChanged(MappingModel mappingModel)
{
this.MappingChanged?.Invoke(mappingModel);
}


#region Methods
//Temp Helper method
public bool GetIsSEDPin()
```

```
{
bool isSEDPin = false;

List<string> InstrumentValues = this.mappings.Select(m => m.InstrumentName.Split('
')[0]).ToList();
if (!InstrumentValues.Contains("PE32H"))
{
isSEDPin = true;
}

return isSEDPin;
}
#endregion

#region Static Pin Model Methods
public static PinModel GetPinModelFromPin(Pin fromPin)
{
PinModel modelItem = new PinModel();

PinModel.DataSwap(fromPin, modelItem);

return modelItem;
}
public static Pin GetPinFromPinModel(PinModel fromModel)
{
Pin pinItem = new Pin();

PinModel.DataSwap(fromModel, pinItem);

return pinItem;
}



public static void DataSwap(PinModel fromModel, Pin toPin)
{
toPin.PinName = fromModel.PinName;
toPin.PinAlias = fromModel.PinAlias;
toPin.mappings = fromModel.mappings.ToList().ConvertAll(new Converter<MappingModel,
Mapping>(MappingModel.GetMappingFromMappingModel));
}

public static void DataSwap(Pin fromPin, PinModel toModel)
```

```
{
toModel.PinName = fromPin.PinName;
toModel.PinAlias = fromPin.PinAlias;
var tempMappingsList = fromPin.mappings.ConvertAll(new Converter<Mapping,
MappingModel>(MappingModel.GetMappingModelFromMapping));
//toModel.mappings = new ObservableCollection<MappingModel>(tempMappingsList);
foreach(MappingModel mapping in tempMappingsList)
{
//mapping.OnChangeOccured += toModel.MappingChanged;
toModel.mappings.Add(mapping);
}
}
#endregion

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}

}
```

Site.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Models.PinModels
{
/// <summary>
/// A helper class thats used to generate site options from the pin map.
/// (Currently only used for GUI purposes)!
/// </summary>
public class Site
{
```

```
public int SiteID { get; set; }
public string SiteName { get; set; }
}
}
```

Test.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.TestModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace MerlinTestStudio_Demo_Telerik
{
public interface ITest
{
int TestNumber { get; set; }
string TestName { get; set; }
ObservableCollection<TestParameter> Parameters { get; set; }

//Convert to uint !
int TestID { get; }
bool IsTestNumBreak { get; set; }
string TestResult { get; set; }

Test Clone(MerlinProject project);
}

[Serializable]
public class Test : ITest, INotifyPropertyChanged
{
```

```
#region Private Members
private int _testID;
private ObservableCollection<TestParameter> _parameters = new
ObservableCollection<TestParameter>(); // { new TestParameter(), new TestParameter() }
private bool isTestNumBreak = false;
private string _testResult;
#endregion Private Members

#region Public Members
public int TestID
{
get { return _testID; }
set { _testID = value; OnPropertyChanged("TestID"); } //Public set only for deserialization.
}

//Test Parameters
public ObservableCollection<TestParameter> Parameters
{
get { return _parameters; }
set
{
_parameters = value;
OnPropertyChanged("Parameters");
if(value != null && value.Count >= 2)
{
Parameters[0].PropertyChanged += (sender, e) => { OnPropertyChanged("TestNumber"); };
Parameters[1].PropertyChanged += (sender, e) => { OnPropertyChanged("TestName"); };
OnPropertyChanged("TestNumber");
OnPropertyChanged("TestName");
}
}
}

[XmlIgnore]
public int TestNumber //Needs 'Empty collection' protection.
{
get
{
int.TryParse(Parameters[0].Value.ToString(), out int x); //Temporary solution, two way value
binding currently set on Parameter[0].
return x; //If try parse fails it'll return 0 but the value that was set may still be incorrect.
}
```

```
set { Parameters[0].Value = value; OnPropertyChanged("TestNumber"); }
}
[XmlIgnore]
public string TestName //Needs 'Empty collection' protection.
{
get { return Parameters[1].Value.ToString(); }
set { Parameters[1].Value = value; OnPropertyChanged("TestName"); }
}

//public int ID { get; set; }  ///TestID ??
//public uint TestNumber { get; set; }  ///Test Parameter 0
//public string TestName { get; set; }  ///Test Parameter 1
private string _units;
public string Units ///Test Parameter 2 (but not required so may throw error via parameters list on get/set)
{
get { return _units; }
set { _units = value; OnPropertyChanged("Units"); }
}



//For Green Hightlight Row-Display Property.
public bool IsTestNumBreak
{
get { return isTestNumBreak; }
set { isTestNumBreak = value; OnPropertyChanged("IsTestNumBreak"); }
}

//For Sequence Item(s).
public string TestResult
{
get { return _testResult; }
set { _testResult = value; OnPropertyChanged("TestResult"); }
}

//Commented out : Not correct structure.
#region Test Limit Properties
//private double _fTlower;
//public double FTLower
//{
//    get { return _fTlower; }
//    set { _fTlower = value; OnPropertyChanged("FTLower"); }
```

```
//}
//private double _fTUpper;
//public double FTUpper
//{
//    get { return _fTUpper; }
//    set { _fTUpper = value; OnPropertyChanged("FTUpper"); }
//}
//private double _qALower;
//public double QALower
//{
//    get { return _qALower; }
//    set { _qALower = value; OnPropertyChanged("QALower"); }
//}
//private double _qAUpper;
//public double QAUpper
//{
//    get { return _qAUpper; }
//    set { _qAUpper = value; OnPropertyChanged("QAUpper"); }
//}
//
//
//private double _goldRetestLower;
//public double GoldRetestLower
//{
//    get { return _goldRetestLower; }
//    set { _goldRetestLower = value; OnPropertyChanged("GoldRetestLower"); }
//}
//private double _goldRetestUpper;
//public double GoldRetestUpper
//{
//    get { return _goldRetestUpper; }
//    set { _goldRetestUpper = value; OnPropertyChanged("GoldRetestUpper"); }
//}
//private GoldControl _goldRetestControl;
//public GoldControl GoldRetestControl
//{
//    get { return _goldRetestControl; }
//    set { _goldRetestControl = value; OnPropertyChanged("GoldRetestControl"); }
//}
//private double _goldTestLower;
//public double GoldTestLower
//{
```

```
//    get { return _goldTestLower; }
//    set { _goldTestLower = value; OnPropertyChanged("GoldTestLower"); }
//}
//private double _goldTestUpper;
//public double GoldTestUpper
//{
//    get { return _goldTestUpper; }
//    set { _goldTestUpper = value; OnPropertyChanged("GoldTestUpper"); }
//}
//private int _GoldTestControl;
//public int GoldTestControl
//{
//    get { return _GoldTestControl; }
//    set { _GoldTestControl = value; OnPropertyChanged("GoldTestControl"); }
//}
//
//private double _GoldLinearLower;
//public double GoldLinearLower
//{
//    get { return _GoldLinearLower; }
//    set { _GoldLinearLower = value; OnPropertyChanged("GoldLinearLower"); }
//}
//
//private double _GoldLinearUpper;
//public double GoldLinearUpper
//{
//    get { return _GoldLinearUpper; }
//    set { _GoldLinearUpper = value; OnPropertyChanged("GoldLinearUpper"); }
//}
//
//private int _GoldLinearControl;
//public int GoldLinearControl
//{
//    get { return _GoldLinearControl; }
//    set { _GoldLinearControl = value; OnPropertyChanged("GoldLinearControl"); }
//}
//
//public bool ApplyAfter { get; set; } //Replaced by ApplyOffset Enum. Separate property created for backwards compatible loading.
//
//private ObservableCollection<double> _offsetSites;
//public ObservableCollection<double> OffsetSites
```

```
//{
//    get { return _offsetSites; }
//    set { _offsetSites = value; OnPropertyChanged("OffsetSites"); }
//}
//private OffsetControl _applyOffset;
//public OffsetControl ApplyOffset
//{
//    get { return _applyOffset; }
//    set { _applyOffset = value; OnPropertyChanged("ApplyOffset"); }
//}
//
//private int _hardBinNumber;
//public int HardBinNumber
//{
//    get { return _hardBinNumber; }
//    set { _hardBinNumber = value; OnPropertyChanged("HardBinNumber"); }
//}
//private string _hardBinName;
//public string HardBinName
//{
//    get { return _hardBinName; }
//    set { _hardBinName = value; OnPropertyChanged("HardBinName"); }
//}
//private string _hardBinPF;
//public string HardBinPF
//{
//    get { return _hardBinPF; }
//    set { _hardBinPF = value; OnPropertyChanged("HardBinPF"); }
//}
//private int _softBinNumber;
//public int SoftBinNumber
//{
//    get { return _softBinNumber; }
//    set { _softBinNumber = value; OnPropertyChanged("SoftBinNumber"); }
//}
//private string _softBinName;
//public string SoftBinName
//{
//    get { return _softBinName; }
//    set { _softBinName = value; OnPropertyChanged("SoftBinName"); }
//}
//private string _softBinPF;
```

```
//public string SoftBinPF
//{
//    get { return _softBinPF; }
//    set { _softBinPF = value; OnPropertyChanged("SoftBinPF"); }
//}
//private ObservableCollection<MultiPassBin> _multiPassBins = new
ObservableCollection<MultiPassBin>();
//public ObservableCollection<MultiPassBin> MultiPassBins
//{
//    get { return _multiPassBins; }
//    set { _multiPassBins = value; OnPropertyChanged("MultiPassBins"); }
//}
//private ObservableCollection<MultiFailBin> _multiFailBins = new
ObservableCollection<MultiFailBin>();
//public ObservableCollection<MultiFailBin> MultiFailBins
//{
//    get { return _multiFailBins; }
//    set { _multiFailBins = value; OnPropertyChanged("MultiFailBins"); }
//}
//
//
//private OffsetControl _applyTraceLoss;
//public OffsetControl ApplyTraceLoss
//{
//    get { return _applyTraceLoss; }
//    set { _applyTraceLoss = value; OnPropertyChanged("ApplyTraceLoss"); }
//}
//
//public ObservableCollection<double> TracelossSites = new ObservableCollection<double>();
//
//public ObservableCollection<double> TestFixturesSites = new ObservableCollection<double>();

#endregion Test Limit Properties

#endregion Public Members

#region Constructor
public Test() // Constructor for deserializing.
{
////Parameters = new ObservableCollection<TestParameter>() { new TestParameter(), new
TestParameter() };
//Parameters[0].PropertyChanged += (sender, e) => { OnPropertyChanged("TestNumber"); };
```

```
//Parameters[1].PropertyChanged += (sender, e) => { OnPropertyChanged("TestName"); };
}

public Test(int testID) //Constructor for newly created tests during runtime.
{
this.TestID = testID;
Parameters = new ObservableCollection<TestParameter>() { new TestParameter(), new
TestParameter() };
}
#endregion Constructor

#region Methods
public Test Clone(MerlinProject project)
{
var clone = new Test(testID: project.GetNewTestID())
{
TestNumber = this.TestNumber,
TestName = this.TestName,
Parameters = new ObservableCollection<TestParameter>(this.Parameters.Select(c =>
c.Clone()).ToList()),
IsTestNumBreak = this.IsTestNumBreak,
TestResult = this.TestResult
};

return clone;
}


//public static void LoadAllFromProgramTestLimit(string FileName, out UpperLowerLimitsData
limitsAndBinnigsData, out GoldLimitsData goldsData, out OffsetLimitsData offsetsData)
//{
//    try
//    {
//        throw new NotImplementedException("This method needs to be re-written to read in the
data from separate files.");
//        #region changes the extension to xml
//        string originalExtension = Path.GetExtension(FileName);
//        //string xmlFilePath = Path.ChangeExtension(FileName, ".xml");
//
//        //if Original extension is CSV, then use the CSV load method, if anything else load using xml
method.
//        var programTestLimit = originalExtension == ".csv" ?
```

```
ProgramTestLimit.ImportCalDataCSV(FileName) :
ProgramTestLimit.ImportCalDataXML(FileName);
//      //use xml file path incase a .csv is loaded in to not overwrite the original data file on Save().
//      //var limitsModel = new TestLimitsDataModel() { FilePath = FileName, LimitFileSource =
programTestLimit, SelectedViewOptional = programTestLimit.ViewOptional };
//      #endregion
//
//      UpperLowerLimitsData limitsData = new UpperLowerLimitsData() { Data = new
List<UpperLowerLimit>() };
//      //Process the xml data from ProgramTestLimit.
//      foreach (var testLimit in programTestLimit.TestLimits)
//      {
//          UpperLowerLimit testToAdd = new UpperLowerLimit()
//          {
//              ID = testLimit.ID,
//              TestNumber = Convert.ToInt32(testLimit.TestNumber),
//              TestName = testLimit.TestName,
//              Units = testLimit.Units,
//              FTLower = testLimit.FTLower,
//              FTUpper = testLimit.FTUpper,
//              QALower = testLimit.QALower,
//              QAUpper = testLimit.QAUpper,
//              HardBinNumber = testLimit.HardBinNumber,
//              HardBinName = testLimit.HardBinName,
//              HardBinPF = testLimit.HardBinPF,
//              SoftBinNumber = testLimit.SoftBinNumber,
//              SoftBinName = testLimit.SoftBinName,
//              SoftBinPF = testLimit.SoftBinPF,
//              MultiPassBins = new ObservableCollection<MultiPassBin>(),
//              MultiFailBins = new ObservableCollection<MultiFailBin>()
//          };
//          limitsData.Data.Add(testToAdd);
//      }
//
//      GoldLimitsData goldLimitsData = new GoldLimitsData() { Data = new List<GoldLimit>() };
//      //Process the xml data from ProgramTestLimit.
//      foreach (var testLimit in programTestLimit.TestLimits)
//      {
//          GoldLimit testToAdd = new GoldLimit()
//          {
//              ID = testLimit.ID,
//              TestNumber = Convert.ToInt32(testLimit.TestNumber),
```

```
//              TestName = testLimit.TestName,
//              Units = testLimit.Units,
//              GoldRetestLower = testLimit.GoldRetestLower,
//              GoldRetestUpper = testLimit.GoldRetestUpper,
//              //GoldRetestControl = testLimit.GoldRetestControl,
//              GoldTestLower = testLimit.GoldTestLower,
//              GoldTestUpper = testLimit.GoldTestUpper,
//              GoldTestControl = testLimit.GoldTestControl,
//              GoldLinearLower = testLimit.GoldLinearLower,
//              GoldLinearUpper = testLimit.GoldLinearUpper,
//              GoldLinearControl = Convert.ToInt32(testLimit.GoldLinearControl)
//          };
//          goldLimitsData.Data.Add(testToAdd);
//      }
//
//      OffsetLimitsData offsetLimitsData = new OffsetLimitsData() { Data = new List<OffsetLimit>()
};
//      foreach (var testLimit in programTestLimit.TestLimits)
//      {
//          List<double> offsets = new List<double>()
//          {
//              testLimit.OffsetSite1,
//              testLimit.OffsetSite2,
//              testLimit.OffsetSite3,
//              testLimit.OffsetSite4,
//              testLimit.OffsetSite5,
//              testLimit.OffsetSite6,
//              testLimit.OffsetSite7,
//              testLimit.OffsetSite8,
//          };
//          OffsetLimit testToAdd = new OffsetLimit()
//          {
//              ID = testLimit.ID,
//              TestNumber = Convert.ToInt32(testLimit.TestNumber),
//              TestName = testLimit.TestName,
//              Units = testLimit.Units,
//              OffsetSites = offsets,
//              //ApplyOffset = testLimit.ApplyOffset,
//          };
//          offsetLimitsData.Data.Add(testToAdd);
//      }
//
```

```
//        //Set the Outputs.
//        limitsAndBinnigsData = limitsData;
//        goldsData = goldLimitsData;
//        offsetsData = offsetLimitsData;
//    }
//    catch (Exception ex)
//    {
//        System.Windows.MessageBox.Show(ex.Message);
//        limitsAndBinnigsData = null;
//        goldsData = null;
//        offsetsData = null;
//    }
//}


#endregion Methods

//Move all static method regions to their correlated VMs, then Delete this class.
#region Static Test & Upper Lower Limit Methods
public static Test GetTestFromUpperLowerLimit(UpperLowerLimit fromLimit)
{
Test testItem = new Test();

DataSwap(fromLimit, testItem);

return testItem;
}
public static UpperLowerLimit GetUpperLowerLimitFromTest(Test fromTest)
{
UpperLowerLimit limitItem = new UpperLowerLimit();

DataSwap(fromTest, limitItem);

return limitItem;
}


/// <summary>
/// Swaps the current limits and binnings data from the Test object to the UpperLowerLimit object.
/// </summary>
/// <param name="fromTest"></param>
/// <param name="toLimit"></param>
```

```
public static void DataSwap(Test fromTest, UpperLowerLimit toLimit)
{
toLimit.ID = fromTest.TestID;
toLimit.TestNumber = Convert.ToInt32(fromTest.TestNumber);
toLimit.TestName = fromTest.TestName;
toLimit.Units = fromTest.Units;
//toLimit.FTLower = fromTest.FTLower;
//toLimit.FTUpper = fromTest.FTUpper;
//toLimit.QALower = fromTest.QALower;
//toLimit.QAUpper = fromTest.QAUpper;
//toLimit.HardBinNumber = fromTest.HardBinNumber;
//toLimit.HardBinName = fromTest.HardBinName;
//toLimit.HardBinPF = fromTest.HardBinPF;
//toLimit.SoftBinNumber = fromTest.SoftBinNumber;
//toLimit.SoftBinName = fromTest.SoftBinName;
//toLimit.SoftBinPF = fromTest.SoftBinPF;
//toLimit.MultiPassBins = fromTest.MultiPassBins;
//toLimit.MultiFailBins = fromTest.MultiFailBins;
}

/// <summary>
/// Swaps the current limits and binnings data from the UpperLowerLimit object to the Test object.
/// </summary>
/// <param name="fromLimit"></param>
/// <param name="toTest"></param>
public static void DataSwap(UpperLowerLimit fromLimit, Test toTest)
{
//toTest.TestID = fromLimit.ID;
toTest.TestNumber = Convert.ToInt32(fromLimit.TestNumber);
toTest.TestName = fromLimit.TestName;
toTest.Units = fromLimit.Units;
//toTest.FTLower = fromLimit.FTLower;
//toTest.FTUpper = fromLimit.FTUpper;
//toTest.QALower = fromLimit.QALower;
//toTest.QAUpper = fromLimit.QAUpper;
//toTest.HardBinNumber = fromLimit.HardBinNumber;
//toTest.HardBinName = fromLimit.HardBinName;
//toTest.HardBinPF = fromLimit.HardBinPF;
//toTest.SoftBinNumber = fromLimit.SoftBinNumber;
//toTest.SoftBinName = fromLimit.SoftBinName;
//toTest.SoftBinPF = fromLimit.SoftBinPF;
//toTest.MultiPassBins = fromLimit.MultiPassBins;
```

```
//toTest.MultiFailBins = fromLimit.MultiFailBins;
}
#endregion Static Test & Upper Lower Limit Methods

#region Static Test & Gold Limit Methods
public static Test GetTestFromGoldLimit(UpperLowerLimit fromLimit)
{
Test testItem = new Test();

DataSwap(fromLimit, testItem);

return testItem;
}
public static GoldLimit GetGoldLimitFromTest(Test fromTest)
{
GoldLimit limitItem = new GoldLimit();

DataSwap(fromTest, limitItem);

return limitItem;
}



/// <summary>
/// Swaps the current golds data from the Test object to the GoldLimit object.
/// </summary>
/// <param name="fromTest"></param>
/// <param name="toLimit"></param>
public static void DataSwap(Test fromTest, GoldLimit toLimit)
{
toLimit.ID = fromTest.TestID;
toLimit.TestNumber = Convert.ToInt32(fromTest.TestNumber);
toLimit.TestName = fromTest.TestName;
toLimit.Units = fromTest.Units;

//toLimit.GoldRetestLower = fromTest.GoldRetestLower;
//toLimit.GoldRetestUpper = fromTest.GoldRetestUpper;
//toLimit.GoldRetestControl = fromTest.GoldRetestControl;
//toLimit.GoldTestLower = fromTest.GoldTestLower;
//toLimit.GoldTestUpper = fromTest.GoldTestUpper;
//toLimit.GoldLinearLower = fromTest.GoldLinearLower;
//toLimit.GoldLinearUpper = fromTest.GoldLinearUpper;
```

```
//toLimit.GoldLinearControl = fromTest.GoldLinearControl;
}

/// <summary>
/// Swaps the current golds data from the GoldLimit object to the Test object.
/// </summary>
/// <param name="fromLimit"></param>
/// <param name="toTest"></param>
public static void DataSwap(GoldLimit fromLimit, Test toTest)
{
//toTest.TestID = fromLimit.ID;
toTest.TestNumber = Convert.ToInt32(fromLimit.TestNumber);
toTest.TestName = fromLimit.TestName;
toTest.Units = fromLimit.Units;

//toTest.GoldRetestLower = fromLimit.GoldRetestLower;
//toTest.GoldRetestUpper = fromLimit.GoldRetestUpper;
//toTest.GoldRetestControl = fromLimit.GoldRetestControl;
//toTest.GoldTestLower = fromLimit.GoldTestLower;
//toTest.GoldTestUpper = fromLimit.GoldTestUpper;
//toTest.GoldTestControl = fromLimit.GoldTestControl;
//toTest.GoldLinearLower = fromLimit.GoldLinearLower;
//toTest.GoldLinearUpper = fromLimit.GoldLinearUpper;
//toTest.GoldLinearControl = fromLimit.GoldLinearControl;

}
#endregion Static Test & Gold Limit Methods

#region Static Test & Offset Limit Methods
public static Test GetTestFromOffsetLimit(OffsetLimit fromLimit)
{
Test testItem = new Test();

DataSwap(fromLimit, testItem);

return testItem;
}
public static OffsetLimit GetOffsetLimitFromTest(Test fromTest)
{
OffsetLimit limitItem = new OffsetLimit();

DataSwap(fromTest, limitItem);
```

```
return limitItem;
}


/// <summary>
/// Swaps the current offsets data from the Test object to the OffsetLimit object.
/// </summary>
/// <param name="fromTest"></param>
/// <param name="toLimit"></param>
public static void DataSwap(Test fromTest, OffsetLimit toLimit)
{
toLimit.ID = fromTest.TestID;
toLimit.TestNumber = Convert.ToInt32(fromTest.TestNumber);
toLimit.TestName = fromTest.TestName;
toLimit.Units = fromTest.Units;

//toLimit.ApplyOffset = fromTest.ApplyOffset;
//toLimit.OffsetSites = fromTest.OffsetSites;
}

/// <summary>
/// Swaps the current offsets data from the OffsetLimit object to the Test object.
/// </summary>
/// <param name="fromLimit"></param>
/// <param name="toTest"></param>
public static void DataSwap(OffsetLimit fromLimit, Test toTest)
{
//toTest.TestID = fromLimit.ID;
toTest.TestNumber = Convert.ToInt32(fromLimit.TestNumber);
toTest.TestName = fromLimit.TestName;
toTest.Units = fromLimit.Units;

//toTest.ApplyOffset = fromLimit.ApplyOffset;
//toTest.OffsetSites = fromLimit.OffsetSites;
}
#endregion Static Test & Offset Limit Methods

#region Static Test & Traceloss Limit Methods

#endregion
```

```
#region Static Test & Test Fixture Limit Methods

#endregion

#region INPC Members

[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}

#endregion
}
}
```

TestParametersDataModel.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Models.TestModels
{
//New Approach... Keep Parameters Static. Make limits insert at 0 index everytime.
public class TestParametersDataModel : INotifyPropertyChanged, IFileData
{
#region Private Members
private string _filePath;
#endregion

public string FileName => Path.GetFileNameWithoutExtension(this.FilePath);
public string FilePath { get { return _filePath; } set { _filePath = value;
OnPropertyChanged("FilePath"); OnPropertyChanged("FileName"); } }
```

```
private ObservableCollection<ITest> _testSource = new ObservableCollection<ITest>();
public ObservableCollection<ITest> TestSource
{
get { return _testSource; }
set { _testSource = value; OnPropertyChanged("TestSource"); }
}

#region Events
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;
#endregion Events

#region Constructor
public TestParametersDataModel()
{
//Deserialize constructor.
}
public TestParametersDataModel(string filePath)
{
this.FilePath = filePath;
}
#endregion Constructor

public void Save()
{
CommitEditBeforeSave?.Invoke();

var bf = new BinaryFormatter();
using (FileStream fs = new FileStream(Path.Combine(this.FilePath, "TestItemsData.bin"),
FileMode.Create, FileAccess.ReadWrite))
bf.Serialize(fs, this.TestSource);
//throw new NotImplementedException();
}

public void Load()
{
//throw new NotImplementedException();
}

public bool Undo()
{
```

```
//throw new NotImplementedException();
return false;
}

public bool Redo()
{
//throw new NotImplementedException();
return false;
}

#region INPC Members
[field: NonSerialized]
public event PropertyChangedEventHandler PropertyChanged;
private void OnPropertyChanged(string propertyName)
{
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
OnChangeOccured();
}
#endregion

public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
}
}
```

DUTSequenceItemSelector.cs

```
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.Data.DataTemplateSelectors
{
public class DUTSequenceItemSelector : DataTemplateSelector
{
public override DataTemplate SelectTemplate(object item, DependencyObject container)
{
FrameworkElement element = container as FrameworkElement;

ISequenceItemComponent Item = item as ISequenceItemComponent;
if (Item.ItemType == SequenceItemType.Function)
```

```
{
return element.FindResource("FunctionItemTemplate") as DataTemplate;
}
else if(Item.ItemType == SequenceItemType.DataControl)
{
return element.FindResource("SequenceControlTemplate") as DataTemplate;
}
else { return null; }
}
}
}
```

ParameterTemplateSelector.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.Data.Selectors.DataTemplateSelectors
{
class ParameterTemplateSelector : DataTemplateSelector
{
public override DataTemplate SelectTemplate(object item, DependencyObject container)
{
FrameworkElement element = container as FrameworkElement;

Parameter param = item as Parameter;
switch (param.Pointer)
{
case MappingPointer.Parameter:
return element.FindResource("ParameterPointerTemplate") as DataTemplate;
case MappingPointer.Pin_Parameter:
return element.FindResource("PinParameterPointerTemplate") as DataTemplate;
case MappingPointer.Manual:
return element.FindResource("ManualPointerTemplate") as DataTemplate;
default:
return null;
}
```

```
}
}

}
```

DUT_TestRowStyleSelector.cs

```
using System.Windows;
using System.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.Data.StyleSelectors
{
public class DUT_TestRowStyleSelector : StyleSelector
{
Style BaseStyle = Application.Current.FindResource("NormalRowStyle") as Style;
Style HaltStyle = Application.Current.FindResource("HaltStyle") as Style;
Style EnabledStyle = Application.Current.FindResource("EnabledStyle") as Style;
Style DisabledStyle = Application.Current.FindResource("DisabledStyle") as Style;

public override Style SelectStyle(object item, DependencyObject container)
{
if (item is ISequenceItem)
{
ISequenceItem group = item as ISequenceItem;

switch (group.State)
{
case "Normal":
return BaseStyle;
case "Enabled":
return EnabledStyle;
case "Halted":
return HaltStyle;
case "Disabled":
return DisabledStyle;
default:
break;
}
}
return null;
}
}
}
```

ProjectTreeViewItemStyleSelector.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.Data.Selectors.StyleSelectors
{
public class ProjectTreeViewItemStyleSelector : StyleSelector
{
public override System.Windows.Style SelectStyle(object item,
System.Windows.DependencyObject container)
{
if (item is MerlinProject)
return this.ProjectStyle;
else if (item is ProjectFolder)
return this.DirectoryStyle;
else if (item is PaneViewModel || item is PVM)
return this.FileStyle;
return null;
}

public Style DirectoryStyle
{
get;
set;
}
public Style FileStyle
{
get;
set;
}
public Style ProjectStyle
{
get;
set;
}
```

```
}
}
```

TestStyleSelector.cs

```
using System.Windows;
using System.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.Data.Selectors.StyleSelectors
{
public class TestStyleSelector : StyleSelector
{
Style BaseStyle = Application.Current.FindResource("TestBaseStyle") as Style;
Style BreakStyle = Application.Current.FindResource("TestBreakStyle") as Style;

public override Style SelectStyle(object item, DependencyObject container)
{
if (item is ITest)
{
ITest test = item as ITest;

switch (test.IsTestNumBreak)
{
case true:
return BreakStyle;
case false:
return BaseStyle;
default:
break;
}
}
return null;
}

}
}
```

DataFormattingService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Services
{
public interface IDataFormattingService
{
/// <summary>
/// Transfers the extracted data from the DFT to the correct DataTable
/// </summary>
void TransferExtractedData();

}

public class DataFormattingService : IDataFormattingService
{


public DataFormattingService()
{

}

public void TransferExtractedData()
{
throw new NotImplementedException();
}
}
}
```

ExtensionService.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

namespace MerlinTestStudio_Demo_Telerik.Data.Services
{
public interface IExtensionService
```

```
{
/// <summary>
/// Imports information about the assembly types contained within a .dll file using
System.Reflection
/// </summary>
/// <param name="assemblyFilePath"></param>
/// <returns></returns>
ObservableCollection<Data.DLL> ImportAssmblyTypes(string assemblyFilePath);

//Implement into UI.
void DeleteDLL();

/// <summary>
/// Returns the collection of DLLs from the data store.
/// </summary>
/// <returns></returns>
ObservableCollection<Data.DLL> GetAllDLLs();
}

public class ExtensionService : IExtensionService
{
/// <summary>
/// ExtensionService Constructor.
/// </summary>
public ExtensionService()
{

}


public ObservableCollection<DLL> ImportAssmblyTypes(string assemblyFilePath)
{
throw new NotImplementedException();
//var types = new ObservableCollection<Data.DLL>();
//
//try
//{
//    Assembly asm = Helpers.AssemblyLoader.LoadWithDependencies(assemblyFilePath);
//
//    ObservableCollection<Data.DLL> DLL_Types = new ObservableCollection<Data.DLL>();
//
//    //Command now begins to start the creation of the DLL based off of the .dll
```

```
//    foreach (Type type in asm.GetTypes())
//    {
//        if (type.IsPublic == true && type.IsEnum != true)
//        {
//            //Gets and then sets the added functions.
//            var FunctionLibrary = new ObservableCollection<ISequenceItemComponent>();
//
//            foreach (MethodInfo MI in type.GetMethods(BindingFlags.Instance | BindingFlags.Public |
BindingFlags.DeclaredOnly))
//            {
//                if (MI.IsSpecialName != true && MI.IsVirtual != true)
//                {
//                    //Gets the parameters of the method(s) inside the .dll
//                    var ParamList = new ObservableCollection<Data.Parameter>();
//
//                    foreach (ParameterInfo PI in MI.GetParameters())
//                    {
//                        Data.Parameter newParam = new Data.Parameter() { ParamName = PI.Name,
ParamType = PI.ParameterType.ToString() };
//
//                        if (PI.ParameterType.IsEnum == true)
//                        {
//                            newParam.IsEnum = true;
//                            newParam.ParamOptions =
GetEnumValuesAsStrings(PI.ParameterType.GetEnumValues());
//                            newParam.Pointer = MappingPointer.Parameter;
//                        }
//                        else
//                        {
//                            newParam.IsEnum = false;
//                            newParam.Pointer = MappingPointer.Parameter;
//                        }
//
//                        ParamList.Add(newParam);
//                    }
//
//                    List<string> resultOptions = new List<string>();
//                    if(MI.ReturnType.Name != "Void" && MI.ReturnType.Namespace == "System") //If
the method return type is not System.Void
//                        resultOptions.Add($"{MI.Name}:{MI.ReturnType.Name}");
//
//                    //Adds each field name of the MethodInfo.ReturnType if it is not included in the
```

```
'System' namespace.
//                  if (MI.ReturnType.Namespace != "System")
//                      foreach (FieldInfo f_info in MI.ReturnType.GetFields(BindingFlags.Instance |
BindingFlags.Public | BindingFlags.DeclaredOnly))
//                          resultOptions.Add($"{MI.Name}:{f_info.Name}");
//
//                  FunctionLibrary.Add(new Data.Function()
//                  {
//                      FunctionName = MI.Name,
//                      DLLFilePath = assemblyFilePath,
//                      DLLClassType = type.ToString(),
//                      NameOfDLL = System.IO.Path.GetFileName(assemblyFilePath),
//                      FunctionParameters = ParamList,
//                      FunctionResultOptions = resultOptions
//                  });
//              }
//          }
//
//          DLL_Types.Add(new Data.DLL()
//          {
//              NameOfDLL = type.ToString(),
//              DLLFilePath = assemblyFilePath,
//              DLLFunctions = FunctionLibrary
//          });
//      }
//  }
//  foreach (Data.DLL dll in DLL_Types)
//      types.Add(dll);
//}
//catch (Exception Ex)
//{
//  string errorMessage = "";
//
//  if (Ex is ReflectionTypeLoadException)
//      foreach (Exception e in (Ex as ReflectionTypeLoadException).LoaderExceptions)
//          errorMessage += string.Format("\n{0}", e.Message);
//  else errorMessage = Ex.Message;
//
//  MessageBox.Show(errorMessage);
//  return types;
//}
//
```

```
//return types;
}

private static List<string> GetEnumValuesAsStrings(Array enum_obj)
{
throw new NotImplementedException();
//List<string> enum_values = new List<string>();
//if (enum_obj == null) { return enum_values; } // Temporary fix, remove later
//foreach (var enumVal in enum_obj)
//    enum_values.Add($"{enumVal}");
//
//return enum_values;
}

public ObservableCollection<DLL> GetAllDLLs()
{
throw new NotImplementedException();
//return DataStorage._allDLLs;
}

public void DeleteDLL()
{
throw new NotImplementedException();
}
}
}
```

InstrumentService.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Services
{
public interface IInstrumentService
{
ObservableCollection<Data.Instrument> GetAllInstruments();
}
```

```
public class InstrumentService : IInstrumentService
{
public InstrumentService()
{

}

public ObservableCollection<Instrument> GetAllInstruments()
{
throw new NotImplementedException();
//return DataStorage._allInstruments;
}
}
}
```

ParameterValueService.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using UnitConversionLib;

namespace MerlinTestStudio_Demo_Telerik.Data.Services
{
public interface IParameterValueService
{
/// <summary>
/// Tunnels to the parameter data to read or write information from the data.
/// </summary>
/// <param name="group"></param>
/// <param name="parameter"></param>
/// <param name="ReadOrWrite"></param>
void TunnelToData(ISequenceItem group, Parameter parameter, string ReadOrWrite);

void WriteToTestParameters(ISequenceItem si, Parameter parameter);
}

public class ParameterValueService : IParameterValueService
```

```
{

public void WriteToTestParameters(ISequenceItem si, Parameter parameter)
{
throw new NotImplementedException();
//try
//{
//    object ProcessedValue = string.Empty; //Can be the value being read from or written to a test.
//    int valueIndex = DataStorage._allParameterMapTags.IndexOf(parameter.ParamMapping);
//    bool isGroup = si is IGroupable;
//
//    //Can't get or set anything with out a mapping.
//    if (parameter.ParamMapping == null) { parameter.TempParamMappingValue = "Not Mapped";
return; }
//
//    //Process the value and determine if it needs to be modified.
//    if (!parameter.IsEnum && parameter.ParamMapping.IsUnitable)
//    {
//        string modifiedValue;
//
Data.Services.UnitService.ValueUnitModifier(parameter.TempParamMappingValue.ToString(),
parameter.ParamUnit, parameter.ParamMapping.Unit, out modifiedValue);
//        ProcessedValue = modifiedValue;
//    }
//    else
//    {
//        ProcessedValue = parameter.TempParamMappingValue;
//    }
//
//    if (!isGroup) //Handle a UnGroupedTest
//        switch (parameter.Pointer)
//        {
//            case MappingPointer.Parameter:
//                if (parameter.TempParamMappingValue != null)
//                {
//                    (si as UnGroupedTest).Test.Parameters[valueIndex].Value = ProcessedValue;
//                }
//                break;
//            case MappingPointer.Pin_Parameter:
//                (si as UnGroupedTest).Test.Parameters[valueIndex].Value =
parameter.TempParamMappingValue;
//                break;
```

```
//            default: break; //case MappingPointer.Manual: //Value Data Bound.
//        }
//    else if (isGroup) //Handle a TestGroup.
//        switch (parameter.Pointer)
//        {
//            case MappingPointer.Parameter:
//                if (parameter.TempParamMappingValue != null)
//                {
//                    foreach (ITest t in (si as TestGroup).Tests)
//                        t.Parameters[valueIndex].Value = ProcessedValue;
//                }
//                break;
//            case MappingPointer.Pin_Parameter:
//                if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the
group to read from.
//                    foreach (ITest t in (si as TestGroup).Tests)
//                        t.Parameters[valueIndex].Value = parameter.TempParamMappingValue;
//                break;
//            default: break; //case MappingPointer.Manual: //Value Data Bound.
//        }
//
//}
//catch(Exception ex) { MessageBox.Show(ex.Message); return; }
}

//Not in use.
public void ReadFromTestParameters(ISequenceItem si, Parameter parameter)
{
throw new NotImplementedException();
//try
//{
//    string modifiedValue;
//    object ProcessedValue = string.Empty; //Can be the value being read from or written to a test.
//    int valueIndex = DataStorage._allParameterMapTags.IndexOf(parameter.ParamMapping);
//
//    bool isGroup = si is IGroupable;
//
//    if (parameter.ParamMapping == null)
//    {
//        parameter.TempParamMappingValue = "Not Mapped";
//        return; //Can't get or set anything with out a mapping.
//    }
```

```
//
//    switch (parameter.Pointer)
//    {
//        case MappingPointer.Manual: //Value Data Bound.
//            break;
//        case MappingPointer.Parameter:
//            if (si is UnGroupedTest)
//                ProcessedValue = (si as UnGroupedTest).Test.Parameters[valueIndex].Value == null ?
string.Empty : (si as UnGroupedTest).Test.Parameters[valueIndex].Value.ToString();
//            else if (si is TestGroup)
//                if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the
group to read from.
//                    ProcessedValue = (si as TestGroup).Tests[0].Parameters[valueIndex].Value == null
? string.Empty : (si as TestGroup).Tests[0].Parameters[valueIndex].Value.ToString();
//
//            //Process the value and determine if it needs to be modified.
//            if (!parameter.IsEnum && parameter.ParamMapping.IsUnitable)
//            {
//                Data.Services.UnitService.ValueUnitModifier(ProcessedValue.ToString(),
parameter.ParamMapping.Unit, parameter.ParamUnit, out modifiedValue);
//                parameter.TempParamMappingValue = modifiedValue;
//            }
//            else
//            {
//                parameter.TempParamMappingValue = ProcessedValue;
//            }
//            break;
//        case MappingPointer.Pin_Parameter:
//
//            string pinName = string.Empty;
//            string result = string.Empty;
//
//            if (si is UnGroupedTest)
//                pinName = (si as UnGroupedTest).Test.Parameters[valueIndex].Value == null ?
string.Empty : (si as UnGroupedTest).Test.Parameters[valueIndex].Value.ToString();
//            else if (si is TestGroup)
//                if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the
group to read from.
//                    pinName = (si as TestGroup).Tests[0].Parameters[valueIndex].Value == null ?
string.Empty : (si as TestGroup).Tests[0].Parameters[valueIndex].Value.ToString();
//
//            foreach (Pin p in DataStorage._allPins)
```

```
//                if (p.PinName == pinName)
//                    foreach (Mapping map in p.mappings)
//                        result += string.Format(" {0}:{1} ,", map.InstrumentName, map.IO);
//
//            parameter.TempParamMappingValue = pinName;
//            parameter.Value = result;
//            break;
//    }
//
//}
//catch (Exception ex) { MessageBox.Show(ex.Message); return; }
}


public void TunnelToData(ISequenceItem si, Parameter parameter, string ReadOrWrite)
{
throw new NotImplementedException();
//try
//{
//    string modifiedValue;
//    object ProcessedValue = string.Empty; //Can be the value being read from or written to a test.
//    int valueIndex = DataStorage._allParameterMapTags.IndexOf(parameter.ParamMapping);
//
//    bool isGroup = si is IGroupable;
//
//    if (parameter.ParamMapping == null)
//    {
//        parameter.TempParamMappingValue = "Not Mapped";
//        return; //Can't get or set anything with out a mapping.
//    }
//
//    switch (ReadOrWrite)
//    {
//        case "WRITE":
//
//            //Process the value and determine if it needs to be modified.
//            if (!parameter.IsEnum && parameter.ParamMapping.IsUnitable)
//            {
//
Data.Services.UnitService.ValueUnitModifier(parameter.TempParamMappingValue.ToString(),
parameter.ParamUnit, parameter.ParamMapping.Unit, out modifiedValue);
//                ProcessedValue = modifiedValue;
```

```
//          }
//          else
//          {
//              ProcessedValue = parameter.TempParamMappingValue;
//          }
//
//          switch (parameter.Pointer)
//          {
//              case MappingPointer.Manual: //Value Data Bound.
//                  break;
//              case MappingPointer.Parameter:
//                  if (parameter.TempParamMappingValue != null)
//                  {
//                      if (si is UnGroupedTest)
//                          (si as UnGroupedTest).Test.Parameters[valueIndex].Value = ProcessedValue;
//                      else if (si is TestGroup)
//                          foreach(ITest t in (si as TestGroup).Tests)
//                              t.Parameters[valueIndex].Value = ProcessedValue;
//                  }
//                  break;
//              case MappingPointer.Pin_Parameter:
//                  if (si is UnGroupedTest)
//                      (si as UnGroupedTest).Test.Parameters[valueIndex].Value =
parameter.TempParamMappingValue;
//                  else if (si is TestGroup)
//                      if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the
group to read from.
//                          foreach(ITest t in (si as TestGroup).Tests)
//                              t.Parameters[valueIndex].Value = parameter.TempParamMappingValue;
//                  break;
//          }
//          break;
//      case "READ":
//          switch (parameter.Pointer)
//          {
//              case MappingPointer.Manual: //Value Data Bound.
//                  break;
//              case MappingPointer.Parameter:
//                  if (si is UnGroupedTest)
//                      ProcessedValue = (si as UnGroupedTest).Test.Parameters[valueIndex].Value ==
null ? string.Empty : (si as UnGroupedTest).Test.Parameters[valueIndex].Value.ToString();
//                  else if (si is TestGroup)
```

```
//                    if((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the group to read from.
//                        ProcessedValue = (si as TestGroup).Tests[0].Parameters[valueIndex].Value == null ? string.Empty : (si as TestGroup).Tests[0].Parameters[valueIndex].Value.ToString();
//
//                //Process the value and determine if it needs to be modified.
//                if (!parameter.IsEnum && parameter.ParamMapping.IsUnitable)
//                {
//                    Data.Services.UnitService.ValueUnitModifier(ProcessedValue.ToString(), parameter.ParamMapping.Unit, parameter.ParamUnit, out modifiedValue);
//                    parameter.TempParamMappingValue = modifiedValue;
//                }
//                else
//                {
//                    parameter.TempParamMappingValue = ProcessedValue;
//                }
//                break;
//            case MappingPointer.Pin_Parameter:
//
//                string pinName = string.Empty;
//                string result = string.Empty;
//
//                if (si is UnGroupedTest)
//                    pinName = (si as UnGroupedTest).Test.Parameters[valueIndex].Value == null ? string.Empty : (si as UnGroupedTest).Test.Parameters[valueIndex].Value.ToString();
//                else if (si is TestGroup)
//                    if ((si as TestGroup).Tests.Count > 0) //Makes sure there is at least one test in the group to read from.
//                        pinName = (si as TestGroup).Tests[0].Parameters[valueIndex].Value == null ? string.Empty : (si as TestGroup).Tests[0].Parameters[valueIndex].Value.ToString();
//
//                foreach (Pin p in DataStorage._allPins)
//                    if (p.PinName == pinName)
//                        foreach (Mapping map in p.mappings)
//                            result += string.Format(" {0}:{1} ,", map.InstrumentName, map.IO);
//
//                parameter.TempParamMappingValue = pinName;
//                parameter.Value = result;
//                break;
//        }
//        break;
//    }
```

```
//}
//catch(Exception ex) { MessageBox.Show(ex.Message); return; }
}


public void CheckGroupTestParametersMatch()
{
throw new NotImplementedException();
}



}
}
```

PinMapService.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Services
{
public interface IPinMapService
{
ObservableCollection<Data.PinModel> GetAllPins();
}

public class PinMapService : IPinMapService
{
public PinMapService()
{

}


public ObservableCollection<PinModel> GetAllPins()
{
throw new NotImplementedException();
//return DataStorage._allPins;
}
```

```
}
}
```

ProjectConfigurationService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Services
{
public interface IProjectConfigurationService
{
///Setup the insturments for the user's selected System Target.
void SetupUserSystemTarget(string SystemTarget);

}

public class ProjectConfigurationService : IProjectConfigurationService
{
//private List<string> _targetSystemOptions = new List<string>() { "APS-500", "APS-300", "APS-
100" };
//public IEnumerable<string> TargetSystemOptions { get { return _targetSystemOptions; } }
//public string UserTargetSystem { get; set; }


public ProjectConfigurationService()
{

}

public void SetupUserSystemTarget(string SystemTarget)
{
throw new NotImplementedException();
//DataStorage._allInstruments.Clear();
//switch (SystemTarget)
//{
//    case "APS-500":
//    case "APS-300":
```

```
//        DataStorage._allInstruments.Add(new Data.Instrument() { InstrumentName = "RF110",
InstrumentID = 0 });
//        DataStorage._allInstruments.Add(new Data.Instrument() { InstrumentName = "RF210",
InstrumentID = 1 });
//        break;
//    case "APS-100":
//        DataStorage._allInstruments.Add(new Data.Instrument() { InstrumentName = "RF100",
InstrumentID = 0 });
//        DataStorage._allInstruments.Add(new Data.Instrument() { InstrumentName = "RF200",
InstrumentID = 1 });
//        break;
//    default:
//        break;
//}

}

}
}
```

SequenceService.cs

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.Xml;
using System.Linq;
using System.IO;
using System;
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Services;
using Telerik.Windows.Documents.Spreadsheet.Expressions.Functions;
using System.Windows;

namespace MerlinTestStudio_Demo_Telerik
{
public interface ISequenceService
{

void ReorderSequenceNumbers();

/// <summary>
```

```
/// Builds the SequenceItems from Tests.
/// </summary>
void BuildSequenceCollection();

/// <summary>
/// Compiles the sequence into a XML file using data provided in the test collection.
/// </summary>
/// <param name="filePath"></param>
void CompileTestSequence(string filePath);

//Formerly known as "GroupAttributeCheck".
/// <summary>
/// Checks if all parameters in each group are the same
/// </summary>
void AllTestGroupParametersCheck();

/// <summary>
/// Checks the parameters of a single TestGroup.
/// </summary>
/// <param name="testGroup"></param>
void TestGroupParametersCheck(TestGroup testGroup);

/// <summary>
/// Retrieves the Group objects in the data store.
/// </summary>
/// <returns></returns>
ObservableCollection<ISequenceItem> GetAllGroups();
}

public class SequenceService : ISequenceService
{
//public static ObservableCollection<string> Sequences = new ObservableCollection<string>();



#region Constructor

IParameterValueService _parameterValueService;
public SequenceService(IParameterValueService parameterValueService)
{
_parameterValueService = parameterValueService;
}
```

```
#endregion

public void ReorderSequenceNumbers()
{
throw new NotImplementedException();
//int seqNum = 1;
//
//foreach (ISequenceItem group in DataStorage._allGroups)
//{
//    group.Sequence = seqNum++;
//}
}

//Temp helper method.
public static Function FindAndReturnFunction(string functionName)
{
throw new NotImplementedException();
//foreach (Data.DLL dll in DataStorage._allDLLs)
//{
//    foreach (Data.Function f in dll.DLLFunctions)
//    {
//        if (f.FunctionName == functionName)
//        {
//            return f;
//        }
//    }
//}
//return null;
}

/// <summary>
/// Builds the SequenceItems from Tests.
/// </summary>
public void BuildSequenceCollection()
{
throw new NotImplementedException();
//int sequenceNumber = 1;
//List<string> groups = new List<string>();
//DataStorage._allGroups.Clear();
//
//foreach(ITest test in DataStorage._allTests)
//{
```

```
//    int testnumber = test.TestNumber;
//    string testname = test.TestName;
//    string DefaultCommand = "Execute";
//    string functionCall = test.Parameters[3].Value.ToString();
//    string groupName = test.Parameters[4].Value.ToString();
//    var sequenceComponents = new SequenceCollection<ISequenceItemComponent>();
//
//    //Sets the SequenceComponents according to test function call
//    if (!string.IsNullOrEmpty(functionCall))
//    {
//        var function = FindAndReturnFunction(functionCall);
//
//        if (function != null)
//            sequenceComponents.Add(function);
//        else if (function is null)
//        {
//            Telerik.Windows.Controls.RadWindow SequencingTool = new
Telerik.Windows.Controls.RadWindow()
//            {
//                Header = $"Please match {functionCall} to appropriate function sequence",
//                Content = new UserControls.Tools.FunctionSequencingTool() {
PassedSequenceCollection = sequenceComponents },
//                WindowStartupLocation = System.Windows.WindowStartupLocation.CenterOwner,
//                Width = 650,
//                Height = 450
//            };
//            SequencingTool.ShowDialog();
//        }
//        test.Parameters[3].Value = ""; //Clears value.
//    }
//
//    if (string.IsNullOrEmpty(groupName))
//    {
//        var UG = new UnGroupedTest()
//        {
//            Sequence = sequenceNumber,
//            ItemName = $"Test Number {testnumber} - {testname}",
//            TestName = testname,
//            TestNumber = testnumber,
//            Command = DefaultCommand,
//            SequenceComponents = sequenceComponents,
//            Test = test
```

```
//        };
//
//        DataStorage._allGroups.Add(UG);
//        sequenceNumber++;
//    }
//    else if (!string.IsNullOrEmpty(groupName)) //!groups.Contains(groupName)
//    {
//        if (!groups.Contains(groupName))
//        {
//            var GT = new TestGroup()
//            {
//                Sequence = sequenceNumber,
//                ItemName = groupName,
//                GroupName = groupName,
//                Tests = new List<ITest>() { test },
//                IsExpanded = false,
//                Command = DefaultCommand,
//                SequenceComponents = sequenceComponents
//            };
//
//            groups.Add(groupName); //Add string to list of pre-existing groups.
//            DataStorage._allGroups.Add(GT);
//            sequenceNumber++;
//        }
//        else //Adds the test to the appropriate group.
//        {
//            foreach(ISequenceItem si in DataStorage._allGroups)
//                if(si is IGroupable)
//                    if ((si as IGroupable).GroupName == groupName)
//                        (si as IGroupable).Tests.Add(test);
//        }
//    }
//}
}

//Formerly known as "GroupAttributeCheck".
/// <summary>
/// Checks if all parameters in each group are the same
/// </summary>
public void AllTestGroupParametersCheck()
{
throw new NotImplementedException();
```

```
//foreach (var si in DataStorage._allGroups.Where(si => si is TestGroup))
//{
//    TestGroupParametersCheck(si as TestGroup);
//}
}

/// <summary>
/// Checks the parameters of a single TestGroup.
/// </summary>
/// <param name="testGroup"></param>
public void TestGroupParametersCheck(TestGroup testGroup)
{
throw new NotImplementedException();
//int numOfTests = testGroup.Tests.Count();
//int sharedParams = numOfTests > 0 ? testGroup.Tests[0].Parameters.Count() : 0;
//
//
//for (int i = 5; i < sharedParams; i++)
//{
//    var comparable = testGroup.Tests[0].Parameters[i].Value.ToString();
//    for (int a = 0; a < numOfTests; a++)
//    {
//        if(testGroup.Tests[a].Parameters[i].Value.ToString() != comparable)
//        {
//            MessageBox.Show($"Mismatch in test parameters for TestGroup:
{testGroup.GroupName}");
//        }
//    }
//}
}

//Test method w/ grouped tests handling... May be outdated.
/// <summary>
/// Compiles the sequence into a XML file using data provided in the test collection.
/// </summary>
/// <param name="filePath"></param>
public void CompileTestSequence(string filePath)
{
throw new NotImplementedException();
//using (XmlTextWriter xmlWriter = new XmlTextWriter(filePath, System.Text.Encoding.UTF8) {
Formatting = Formatting.Indented })
//{
```

```
//    xmlWriter.WriteStartDocument();
//    xmlWriter.WriteStartElement("TestSequence");
//    xmlWriter.WriteAttributeString("Version", "1");
//
//    xmlWriter.WriteStartElement("SequenceInformation");
//    xmlWriter.WriteElementString("Product", "VC7643_63H380");
//    xmlWriter.WriteElementString("Revision", "A");
//    xmlWriter.WriteElementString("TestProgram", "VC7643_63H380.dll");
//    xmlWriter.WriteElementString("Comment", "A test program for the VC7643_63H380 product.");
//    xmlWriter.WriteEndElement();
//
//    xmlWriter.WriteComment("Sequence Load Steps exeucute only once at the start of the
sequence even if the sequence file is executed multiple times.");
//    xmlWriter.WriteStartElement("SequenceLoad");
//    xmlWriter.WriteElementString("SequenceLoadStep", "");
//    xmlWriter.WriteEndElement();
//
//    xmlWriter.WriteComment("Sequence UnLoad Steps exeucute only once at the end of the
sequence even if the sequence file is executed multiple times.");
//    xmlWriter.WriteStartElement("SequenceUnLoad");
//    xmlWriter.WriteElementString("SequenceUnLoadStep", "");
//    xmlWriter.WriteEndElement();
//
//    xmlWriter.WriteComment("Sequence Steps execute every time a device is tested.");
//    xmlWriter.WriteStartElement("Sequence");
//    foreach (ISequenceItem SeqItem in DataStorage._allGroups)
//    {
//        if (SeqItem.SequenceComponents.Count > 0 )
//        {
//            foreach (ISequenceItemComponent sic in SeqItem.SequenceComponents)
//            {
//                if (sic.ItemType == SequenceItemType.Function) //Only handles functions and odes
not retrive values from SequenceControls.
//                {
//                    var func = sic as Data.Function;
//                    xmlWriter.WriteStartElement("SequenceStep");
//
//                    ITest test = new Test() { TestNumber = 0, TestResult = string.Empty }; //Make array
for multiple test numbers allowed on sequence step generation!
//                    switch (SeqItem) //Array pos. 0 throw exception when string null maybe.
//                    {
//                        case UnGroupedTest _:
```

```
//                      if ((SeqItem as UnGroupedTest).Test.TestResult.Split(':')[0] ==
func.FunctionName)
//                      {
//                          test = (SeqItem as UnGroupedTest).Test;
//                      }
//                      break;
//                  case TestGroup _:
//                      TestGroup testGroup = (SeqItem as TestGroup);
//                      foreach (var t in testGroup.Tests.Where(t => t.TestResult.Split(':')[0] ==
func.FunctionName))
//                      {
//                          test = t;
//                      }
//                      break;
//              }
//
//              xmlWriter.WriteElementString("TestNumber", test.TestNumber.ToString());
//              xmlWriter.WriteElementString("TestResultMapping", test.TestResult);
//
//              xmlWriter.WriteElementString("DllFilePath", func.DLLFilePath);
//              xmlWriter.WriteElementString("DllClassType", func.DLLClassType);
//              xmlWriter.WriteElementString("MethodName", func.FunctionName);
//              xmlWriter.WriteStartElement("MethodParameters");
//              foreach (Data.Parameter param in (sic as Data.Function).FunctionParameters)
//              {
//                  xmlWriter.WriteStartElement("MethodParameter");
//                  xmlWriter.WriteElementString("Parametername", param.ParamName);
//                  xmlWriter.WriteElementString("Parametertype", param.ParamType.ToString());
//
//                  //Unsure if this handles TestGroup objects well. Check.
//                  _parameterValueService.TunnelToData(SeqItem, param, "READ");
//
//                  xmlWriter.WriteElementString("Parametervalue",
param.TempParamMappingValue.ToString());
//                  xmlWriter.WriteEndElement();
//              }
//              xmlWriter.WriteEndElement();
//              xmlWriter.WriteElementString("PropertName", "");
//              xmlWriter.WriteElementString("PropertType", "");
//              xmlWriter.WriteElementString("PropertValue", "");
//              xmlWriter.WriteEndElement();
//          }
```

```
//            }
//        }
//        else
//        {
//            xmlWriter.WriteStartElement("SequenceStep");
//            xmlWriter.WriteElementString("dll", "");
//            xmlWriter.WriteElementString("MethodName", "");
//            xmlWriter.WriteElementString("MethodParameters", "");
//            xmlWriter.WriteElementString("PropertName", "");
//            xmlWriter.WriteElementString("PropertType", "");
//            xmlWriter.WriteElementString("PropertValue", "");
//            xmlWriter.WriteEndElement();
//        }
//    }
//    xmlWriter.WriteEndElement();
//
//    xmlWriter.WriteEndElement();
//    xmlWriter.WriteEndDocument();
//}
}

public ObservableCollection<ISequenceItem> GetAllGroups()
{
throw new NotImplementedException();
//return DataStorage._allGroups;
}

}

}
```

TestParametersService.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
```

```
namespace MerlinTestStudio_Demo_Telerik.Data.Services
{
public interface ITestParametersService
{
/// <summary>
/// Clears the parameter DataTable's columns, rows, and data.
/// </summary>
void ResetTable();

/// <summary>
/// Adds a new column to the parameter DataTable.
/// </summary>
/// <param name="columnName"></param>
/// <param name="columnDataType"></param>
void AddColumn(string columnName, Type columnDataType, ParameterMapTag paramMapTag);

/// <summary>
/// Gets the test paramaters available (as Pins or Parameters).
/// </summary>
/// <returns></returns>
ObservableCollection<Data.ParameterMapTag> GetTestParameters(string pinsOrParams);

}

public class TestParametersService : ITestParametersService
{
private readonly IViewModelLocator _viewModelLocator;

public TestParametersService(IViewModelLocator viewModelLocator)
{
_viewModelLocator = viewModelLocator;
}

public void AddColumn(string columnName, Type columnDataType, ParameterMapTag
paramMapTag)
{
throw new NotImplementedException();
//try
//{
//    var conditionsVM = _viewModelLocator.GetUsercontrolDataContext("Test Parameters") as
ConditionsViewModel;
//    if (conditionsVM != null)
```

```
//    {
//        //DataStorage.m_CSV_Conditions.Columns.Add(columnName, columnDataType);
//        conditionsVM.AddColumn(columnName, paramMapTag.Unit);
//        DataStorage._allParameterMapTags.Add(paramMapTag);
//
//    }
//}
//catch (Exception ex) { MessageBox.Show(ex.Message); }
}

public void ResetTable()
{
throw new NotImplementedException();
//try
//{
//    var conditionsVM = _viewModelLocator.GetUsercontrolDataContext("Test Parameters") as
ConditionsViewModel;
//    if (conditionsVM != null)
//    {
//        //DataStorage.m_CSV_Conditions.Clear();
//        //DataStorage.m_CSV_Conditions.Columns.Clear();
//        conditionsVM.ResetTable(); //Resets the UI elements manually.
//
//        DataStorage._allParameterMapTags.Clear();
//        DataStorage._allTests.Clear();
//    }
//}
//catch (Exception ex) { MessageBox.Show(ex.Message); }
}

public ObservableCollection<Data.ParameterMapTag> GetTestParameters(string pinsOrParams)
{
throw new NotImplementedException();
//return DataStorage._allParameterMapTags;
////int count = DataStorage._allParameterMapTags.Count;
////switch (pinsOrParams)
////{
////    case "Pins":
////        //var pins = new ObservableCollection<ParameterMapTag>();
////        //for (int i = 12; i < count; i++)
////        //{
////        //    pins.Add(DataStorage._allParameterMapTags[i]);
```

```
////        //}
////        return DataStorage._allParameterMapTags;
////    case "Parameters":
////        //var parameters = new ObservableCollection<ParameterMapTag>();
////        //parameters.Add(DataStorage._allParameterMapTags[0]);
////        //for (int i = 5; i < count; i++)
////        //{
////        //    parameters.Add(DataStorage._allParameterMapTags[i]);
////        //}
////        return DataStorage._allParameterMapTags;
////    default:
////        return DataStorage._allParameterMapTags;
////}
}
}
}
```

TestService.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Data.Services
{
public interface ITestService
{
void RenumerateTestCollection();

void AddNewTest();

void InsertNewTest(int insertIndex);

void RemoveTest(ITest test);

void DuplicateTest(ITest testToClone);

ObservableCollection<ITest> GetTestCollection { get; }
}
```

```
//Service needs error handling.
public class TestService : ITestService
{

public ObservableCollection<ITest> GetTestCollection
{
get
{
throw new NotImplementedException();
//return DataStorage._allTests;
}
}

public void DuplicateTest(ITest testToClone)
{
throw new NotImplementedException();
//var clonedTest = testToClone.Clone();
//int insertIndex = DataStorage._allTests.IndexOf(testToClone);
//DataStorage._allTests.Insert(insertIndex, clonedTest);
//
////Test ID management needed here.
//clonedTest.TestID = DataManagement.GetNewTestID();
//
////handle the test number situation or let the renumeration handle it????
//
//if (string.IsNullOrEmpty(clonedTest.Parameters[4].Value.ToString())) //Must be an
UnGroupedTest.
//{
//    //Find the SequenceItem of the test, duplicate it, and place at the same index.
//    ISequenceItem item = GetSequenceItemOfTest(testToClone);
//    var newUGTest = new UnGroupedTest()
//    {
//        ItemName = $"Test Number {clonedTest.TestNumber} - {clonedTest.TestName}",
//        Test = clonedTest,
//        Sequence = 0,
//        Command = item.Command,
//        State = item.State,
//        SequenceComponents = item.SequenceComponents
//    };
//    DataStorage._allGroups.Add(newUGTest);
//
//}
```

```
//else // Must be a TestGroup.
//{
//    ISequenceItem item = GetSequenceItemOfTest(testToClone);
//
//    // Add the ClonedTest to the TestGroup if group is found.
//    (item as TestGroup).Tests.Add(clonedTest);
//}
}

public void AddNewTest()
{
throw new NotImplementedException();
//ITest newTest = new Test() { TestID = DataManagement.GetNewTestID(), Parameters =
ProvideEmptyParameters() };
//
//DataStorage._allTests.Add(newTest);
//
////Add new UnGroupedTest to the SequenceItem collection.
//ISequenceItem newUGTest = new UnGroupedTest()
//{
//    Sequence = DataStorage._allGroups.Count() + 1,
//    Command = "Execute",
//    SequenceComponents = new SequenceCollection<ISequenceItemComponent>(),
//    Test = newTest
//};
//
//DataStorage._allGroups.Add(newUGTest);
}

public void InsertNewTest(int insertIndex)
{
throw new NotImplementedException();
//ITest newTest = new Test() { TestID = DataManagement.GetNewTestID(), Parameters =
ProvideEmptyParameters() };
//
//DataStorage._allTests.Insert((insertIndex == -1 ? DataStorage._allTests.Count() : insertIndex),
newTest);
//
////Add new UnGroupedTest to the SequenceItem collection.
//ISequenceItem newUGTest = new UnGroupedTest()
//{
//    Sequence = DataStorage._allGroups.Count() + 1,
```

```
//    Command = "Execute",
//    SequenceComponents = new SequenceCollection<ISequenceItemComponent>(),
//    Test = newTest
//};
//
//DataStorage._allGroups.Add(newUGTest);
}

//Switch to using the GetSequenceItemOfTest method.
public void RemoveTest(ITest test)
{
throw new NotImplementedException();
//DataStorage._allTests.Remove(test); //Removes from test collection.
//
////Remove Test from SeqeunceItem collection...
//ISequenceItem UGTest = null;
//
////finds the test if it is in a test group.
//foreach (ISequenceItem si in DataStorage._allGroups)
//{
//    if (si is UnGroupedTest)
//    {
//        if ((si as UnGroupedTest).Test.TestID == test.TestID)
//            UGTest = si;
//    }
//    else if (si is TestGroup)
//    {
//        ITest rTest = (si as TestGroup).Tests.Find(c => c.TestID == test.TestID);
//        if (rTest != null)
//            (si as TestGroup).Tests.Remove(rTest);
//    }
//}
////If an ungrouped test is found remove it.
//if (UGTest != null)
//    DataStorage._allGroups.Remove(UGTest);
}

public void RenumerateTestCollection()
{
throw new NotImplementedException();
//int testnum = 1;
//
```

```
//foreach (ITest t in DataStorage._allTests)
//{
//    if (t.IsTestNumBreak) { testnum = t.TestNumber; } //check if test is designated as a Test
Numeration Break.
//    t.TestNumber = testnum++;
//}
}

#region Private methods

private ObservableCollection<TestParameter> ProvideEmptyParameters()
{
throw new NotImplementedException();
//var testParameters = new ObservableCollection<TestParameter>();
//int maxNumParameters = DataStorage._allParameterMapTags.Count;
//DataStorage._allTests.Max((x) => x.Parameters.Count);
//
////Generate correct number of parameters.
//for (int i = 0; i < maxNumParameters; i++)
//    testParameters.Add(new TestParameter() { Value = string.Empty });
//
//return testParameters;
}

/// <summary>
/// Find and return the SequenceItem in which the given test is associated with.
/// </summary>
/// <param name="test"></param>
/// <returns></returns>
private ISequenceItem GetSequenceItemOfTest(ITest test)
{
throw new NotImplementedException();
//ISequenceItem SeqItem = null;
//
//foreach (ISequenceItem si in DataStorage._allGroups)
//{
//    if (si is UnGroupedTest)
//    {
//        if ((si as UnGroupedTest).Test.TestID == test.TestID)
//            SeqItem = si;
//    }
//    else if (si is TestGroup)
```

```
//    {
//        ITest rTest = (si as TestGroup).Tests.Find(c => c.TestID == test.TestID);
//        if (rTest != null)
//            SeqItem = si;
//    }
//}
//
//return SeqItem;
}

#endregion

}
}
```

UnitService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using UnitConversionLib;

namespace MerlinTestStudio_Demo_Telerik.Data.Services
{
public interface IUnitService
{

}

public class UnitService : IUnitService
{
public readonly List<string> UnitTypes = new List<string>() { "Volts", "Amps", "Frequency",
"Power", "Percentage", "Number" };

public static void MatchUnit(string inputUnit, out string unitType, out string unit)
{
unit = inputUnit;
switch (inputUnit)
{
case "%": unitType = "Percentage"; break;
```

```
case "#": unitType = "Number"; break;
case "dBm":
case "dB":
unitType = "Power";
break;
case "THz":
case "GHz":
case "MHz":
case "KHz":
case "hHz":
case "daHz":
case "Hz":
unitType = "Frequency";
break;
case "V":
case "dV":
case "cV":
case "mV":
case "uV":
case "nV":
case "pV":
unitType = "Volts";
break;
case "A":
case "dA":
case "cA":
case "mA":
case "uA":
case "nA":
case "pA":
unitType = "Amps";
break;
default:
unitType = ""; unit = ""; break;
}
}

public static void ValueUnitModifier(string inputValue, string oldUnit, string newUnit, out string
modifiedValue)
{
modifiedValue = inputValue;
try
```

```
{
if (!string.IsNullOrEmpty(oldUnit) && !string.IsNullOrEmpty(newUnit))
{
double value;
if (!string.IsNullOrEmpty(inputValue))
if (Double.TryParse(inputValue, out value))
{
double ParsableValue = value < 0 ? value * -1 : value; //value needs to go into the
Mesurable.Parse() as a positive number.
double convertedValue = Measurable.Parse($"{ParsableValue}
{oldUnit}").ConvertTo(newUnit).Amount;
modifiedValue = value < 0 ? (convertedValue * -1).ToString() : convertedValue.ToString();
}
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}


}
}
```

.NETFramework,Version=v4.6.1.AssemblyAttributes.cs

```
// <autogenerated />
using System;
using System.Reflection;
[assembly:
global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETFramework,Version=v4.6.1",
FrameworkDisplayName = ".NET Framework 4.6.1")]
```

AssemblyInfo.cs

```
using System.Reflection;
using System.Resources;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.Windows;


// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Merlin Test Studio")]
[assembly: AssemblyDescription("An IDE to develop Merlin Test application code.")]
[assembly: AssemblyConfiguration("")]
```

```
[assembly: AssemblyCompany("Merlin Test Technologies")]
[assembly: AssemblyProduct("")]
[assembly: AssemblyCopyright("Copyright ©  2019")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

//In order to begin building localizable applications, set
//<UICulture>CultureYouAreCodingWith</UICulture> in your .csproj file
//inside a <PropertyGroup>.  For example, if you are using US english
//in your source files, set the <UICulture> to en-US.  Then uncomment
//the NeutralResourceLanguage attribute below.  Update the "en-US" in
//the line below to match the UICulture setting in the project file.

//[assembly: NeutralResourcesLanguage("en-US", UltimateResourceFallbackLocation.Satellite)]


[assembly: ThemeInfo(
ResourceDictionaryLocation.None, //where theme specific resource dictionaries are located
//(used if a resource is not found in the page,
// or application resource dictionaries)
ResourceDictionaryLocation.SourceAssembly //where the generic resource dictionary is located
//(used if a resource is not found in the page,
// app, or any theme specific resource dictionaries)
)]


// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:TempDataTable.Rows[r][DTcolumnIndex] = tag.AttributeType ==
TestAttributeType.Test_Number ? (object)int.Parse(valueToStore) : valueToStore;
// [assembly: AssemblyVersion("0.14.0.01")]
```

```
[assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyFileVersion("0.14.0.01")]
```

Resources.Designer.cs

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace MerlinTestStudio_Demo_Telerik.Properties {
using System;


/// <summary>
///   A strongly-typed resource class, for looking up localized strings, etc.
/// </summary>
// This class was auto-generated by the StronglyTypedResourceBuilder
// class via a tool like ResGen or Visual Studio.
// To add or remove a member, edit your .ResX file then rerun ResGen
// with the /str option, or rebuild your VS project.
[global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyT
ypedResourceBuilder", "16.0.0.0")]
[global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
[global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
internal class Resources {

private static global::System.Resources.ResourceManager resourceMan;

private static global::System.Globalization.CultureInfo resourceCulture;

[global::System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1811:AvoidUncalledPrivateCode")]
internal Resources() {
}

/// <summary>
///   Returns the cached ResourceManager instance used by this class.
```

```
/// </summary>
[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableState.Advanced)]
internal static global::System.Resources.ResourceManager ResourceManager {
get {
if (object.ReferenceEquals(resourceMan, null)) {
global::System.Resources.ResourceManager temp = new global::System.Resources.ResourceManager("MerlinTestStudio_Demo_Telerik.Properties.Resources", typeof(Resources).Assembly);
resourceMan = temp;
}
return resourceMan;
}
}

/// <summary>
///   Overrides the current thread's CurrentUICulture property for all
///   resource lookups using this strongly typed resource class.
/// </summary>
[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableState.Advanced)]
internal static global::System.Globalization.CultureInfo Culture {
get {
return resourceCulture;
}
set {
resourceCulture = value;
}
}
}
}
```

Settings.Designer.cs

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
```

```
namespace MerlinTestStudio_Demo_Telerik.Properties {


[global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.Editors.SettingsDesigner.SettingsSingleFileGenerator", "16.10.0.0")]
internal sealed partial class Settings : global::System.Configuration.ApplicationSettingsBase {

private static Settings defaultInstance =
((Settings)(global::System.Configuration.ApplicationSettingsBase.Synchronized(new Settings())));

public static Settings Default {
get {
return defaultInstance;
}
}
}
}
```

AdditionalContent.cs

```
using System;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows.Controls.Diagrams.Extensions;
using Telerik.Windows.Diagrams.Core;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class AdditionalContent : Control
{
public static readonly DependencyProperty ContextItemProperty =
DependencyProperty.Register("ContextItem", typeof(object), typeof(AdditionalContent), new
PropertyMetadata(null, OnContextItemPropertyChanged));

public static readonly DependencyProperty DiagramProperty =
DependencyProperty.Register("Diagram", typeof(RadDiagram), typeof(AdditionalContent), new
PropertyMetadata(null));
```

```
protected SettingsPane settingsPane;
protected FrameworkElement addRemove;

public RadDiagram Diagram
{
get { return (RadDiagram)GetValue(DiagramProperty); }
set { SetValue(DiagramProperty, value); }
}

public object ContextItem
{
get { return (object)GetValue(ContextItemProperty); }
set { SetValue(ContextItemProperty, value); }
}

static AdditionalContent()
{
CommandManager.RegisterClassCommandBinding(typeof(AdditionalContent), new
CommandBinding(SwimlaneCommands.AddCommand, OnAddCommand));
CommandManager.RegisterClassCommandBinding(typeof(AdditionalContent), new
CommandBinding(SwimlaneCommands.RemoveCommand, OnRemoveCommand,
OnCanExecuteRemove));
}

public override void OnApplyTemplate()
{
base.OnApplyTemplate();

this.settingsPane = this.GetTemplateChild("settingsPane") as SettingsPane;
this.addRemove = this.GetTemplateChild("addRemoveButtons") as FrameworkElement;

if (this.ContextItem != null)
this.OnContextItemChanged(this.ContextItem, null);
}

protected virtual void OnRemove()
{
var container = this.ContextItem as MainContainerShapeBase;
if (container != null && container.OrderedChildren.Count > 0)
{
var itemToRemove = container.OrderedChildren.LastOrDefault();
```

```
this.RemoveContainer(container, itemToRemove as RadDiagramShapeBase);
}
}

protected virtual void OnAdd()
{
var container = this.ContextItem as MainContainerShapeBase;
if (container != null)
{
if (container.ChildrenPositioning == System.Windows.Controls.Orientation.Vertical)
this.AddContainer(container, new RowShape() { ContainerPosition = container.Items.Count });
else
this.AddContainer(container, new TableShape() { ContainerPosition = container.Items.Count });
}
}

protected virtual void OnContextItemChanged(object newValue, object oldValue)
{
if (this.addRemove == null || this.settingsPane == null) return;

if (newValue == null)
{
this.settingsPane.Visibility = Visibility.Collapsed;
this.addRemove.Visibility = Visibility.Collapsed;
this.Visibility = Visibility.Collapsed;
}
else
{
this.Visibility = Visibility.Visible;
if (newValue is MainContainerShapeBase)
this.addRemove.Visibility = Visibility.Visible;
else
this.addRemove.Visibility = Visibility.Collapsed;
this.settingsPane.Visibility = Visibility.Visible;
}
}

protected virtual void OnCanRemove(CanExecuteRoutedEventArgs e)
{
var container = this.ContextItem as MainContainerShapeBase;
if (container != null && container.OrderedChildren != null && container.OrderedChildren.Count > 0)
{
```

```
e.CanExecute = true;
}
}

private static void OnCanExecuteRemove(object sender, CanExecuteRoutedEventArgs e)
{
var addCont = sender as AdditionalContent;
if (addCont != null && addCont.Diagram != null)
{
addCont.OnCanRemove(e);
}
else
{
e.CanExecute = false;
}
}

private static void OnRemoveCommand(object sender, ExecutedRoutedEventArgs e)
{
var addCont = sender as AdditionalContent;
if (addCont != null && addCont.Diagram != null)
{
addCont.OnRemove();
}
}

private static void OnAddCommand(object sender, ExecutedRoutedEventArgs e)
{
var addCont = sender as AdditionalContent;
if (addCont != null && addCont.Diagram != null)
{
addCont.OnAdd();
}
}

private static void OnContextItemPropertyChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
var additionalContent = d as AdditionalContent;
if (additionalContent != null)
{
additionalContent.OnContextItemChanged(e.NewValue, e.OldValue);
```

```
}
}

private void AddContainer(MainContainerShapeBase container, IShape itemToAdd)
{
if (container == null || itemToAdd == null) return;
CompositeCommand command = new CompositeCommand("Add new container");
if (container.IsCollapsed)
{
command.AddCommand(new UndoableDelegateCommand("Update container",
new Action<object>((o) =>
{
container.IsCollapsed = false;
}),
new Action<object>((o) =>
{
container.IsCollapsed = true;
})));
}
command.AddCommand(new UndoableDelegateCommand("Add Container",
new Action<object>((o) =>
{
container.Items.Add(itemToAdd);
container.IsCollapsed = false;
}),
new Action<object>((o) =>
{
this.Diagram.RemoveShape(itemToAdd);
container.IsCollapsed = false;
})));

this.Diagram.UndoRedoService.ExecuteCommand(command);
}

private void RemoveContainer(MainContainerShapeBase container, RadDiagramShapeBase
itemToRemove)
{
if (container == null || itemToRemove == null) return;

CompositeCommand command = new CompositeCommand("Remove horizontal container");
if (container.IsCollapsed)
{
```

```
command.AddCommand(new UndoableDelegateCommand("Update container",
new Action<object>((o) =>
{
container.IsCollapsed = false;
}),
new Action<object>((o) =>
{
container.IsCollapsed = true;
})));
}
command.AddCommand(this.CreateRemoveShapeCommand(itemToRemove));
if (container.IsCollapsed)
{
command.AddCommand(new UndoableDelegateCommand("Update container",
new Action<object>((o) =>
{
}),
new Action<object>((o) =>
{
})));
}

this.Diagram.UndoRedoService.ExecuteCommand(command);
}

private CompositeCommand CreateRemoveShapeCommand(RadDiagramShapeBase shape)
{
if (shape == null) return null;
var compositeRemoveShapeCommand = new
CompositeCommand(CommandNames.RemoveShapes);
var removeCommand = new UndoableDelegateCommand(CommandNames.RemoveShape, s =>
this.Diagram.RemoveShape(shape), s => this.Diagram.AddShape(shape));

var parentContainer = shape.ParentContainer;
if (parentContainer != null)
{
var execute = new Action<object>((o) => parentContainer.RemoveItem(shape));
var undo = new Action<object>((o) =>
{
if (!parentContainer.Items.Contains(shape))
parentContainer.AddItem(shape);
});
```

```
compositeRemoveShapeCommand.AddCommand(new
UndoableDelegateCommand(CommandNames.RemoveItemFromContainer, execute, undo));
}

foreach (var changeSourceCommand in
shape.OutgoingLinks.Union(shape.IncomingLinks).ToList().Select(connection =>
this.CreateRemoveConnectionCommand(connection)))
{
compositeRemoveShapeCommand.AddCommand(changeSourceCommand);
}

compositeRemoveShapeCommand.AddCommand(removeCommand);

var container = shape as RadDiagramContainerShape;
if (container != null)
{
for (int i = container.Items.Count - 1; i >= 0; i--)
{
var shapeToRemove = container.Items[i] as RadDiagramShapeBase;
if (shapeToRemove != null)
compositeRemoveShapeCommand.AddCommand(this.CreateRemoveShapeCommand(shapeTo
Remove));
else
{
var connection = container.Items[i] as IConnection;
if (connection != null && connection.Source == null && connection.Target == null)
compositeRemoveShapeCommand.AddCommand(this.CreateRemoveConnectionCommand(cont
ainer.Items[i] as IConnection));
}
}
}

return compositeRemoveShapeCommand;
}

private UndoableDelegateCommand CreateRemoveConnectionCommand(IConnection
connection)
{
var compositeRemoveConnectionCommand = new
CompositeCommand(CommandNames.RemoveConnections);

UndoableDelegateCommand removeCommand = new
```

```
UndoableDelegateCommand(CommandNames.RemoveConnection, s =>
this.Diagram.RemoveConnection(connection), s => this.Diagram.AddConnection(connection));

compositeRemoveConnectionCommand.AddCommand(removeCommand);

return compositeRemoveConnectionCommand;
}
}
}
```

AttachedProperties.cs

```
using System.Windows;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows.Diagrams.Core;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public static class AttachedProperties
{
public static readonly DependencyProperty CustomConnectorsProperty =
DependencyProperty.RegisterAttached("CustomConnectors", typeof(ConnectorCollection),
typeof(AttachedProperties), new PropertyMetadata(null, OnCustomConnectorsChanged));

public static ConnectorCollection GetCustomConnectors(DependencyObject obj)
{
return (ConnectorCollection)obj.GetValue(CustomConnectorsProperty);
}

public static void SetCustomConnectors(DependencyObject obj, ConnectorCollection value)
{
obj.SetValue(CustomConnectorsProperty, value);
}

private static void OnCustomConnectorsChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
var shape = d as IShape;
if (shape != null && e.NewValue != null)
{
shape.Connectors.Clear();
foreach (RadDiagramConnector item in e.NewValue as ConnectorCollection)
{
```

```
shape.Connectors.Add(new RadDiagramConnector() { Offset = item.Offset, Name = item.Name,
Opacity = item.Opacity });
}
}
}
}
}
```

CustomContainerBase.cs

```
using System;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Input;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows.DragDrop;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class CustomContainerBase : Telerik.Windows.Controls.RadDiagramContainerShape,
Telerik.Windows.Diagrams.Core.IContainerShape
{
private double restoredMinWidth;
private Popup editPopup;
private TextBox editTextBox;

protected Grid headerElement;
protected Grid editHeaderElement;
protected bool templateApplied;
protected bool loaded;
protected double restoredWidth;
protected double actualHeaderWidth;
private bool isRollback;

public bool ShouldUpdateChildren { get; set; }

public Orientation Orientation { get; set; }

public CustomContainerBase()
{
this.Loaded += this.OnCustomContainerLoaded;
```

```
this.ShouldUpdateChildren = true;
}

public override void OnApplyTemplate()
{
var headerPresenter = this.GetTemplateChild("NormalContentPresenter") as FrameworkElement;
if (headerPresenter != null)
headerPresenter.SizeChanged += OnHeaderPresenterSizeChanged;

base.OnApplyTemplate();

if (this.editTextBox != null)
{
this.editTextBox.KeyDown -= OnEditPopupKeyDown;
}

this.headerElement = this.GetTemplateChild("PART_headerElement") as Grid;
this.editHeaderElement = this.GetTemplateChild("PART_editHeaderElement") as Grid;
this.editPopup = this.GetTemplateChild("PART_editPopup") as Popup;
this.editTextBox = this.GetTemplateChild("EditTextBox") as TextBox;

if (this.editPopup != null)
{
this.editPopup.Placement = System.Windows.Controls.Primitives.PlacementMode.Center;
}

if (this.editTextBox != null)
{
this.editTextBox.KeyDown += OnEditPopupKeyDown;
}

this.templateApplied = true;
if (this.loaded)
this.ContentBounds = this.GetNewContentBounds(this.Bounds);
}

public Rect GetNewContentBounds(Rect shapeBounds)
{
if (this.headerElement == null) return shapeBounds;

double margin = Telerik.Windows.Diagrams.Core.DiagramConstants.ContainerMargin;
```

```
if (this.Orientation == Orientation.Vertical)
{
double headerOffset = this.headerElement != null ? this.headerElement.ActualHeight +
this.headerElement.Margin.Top + this.headerElement.Margin.Bottom : 0;

var width = Math.Max(0, shapeBounds.Width - (2 * margin));
var height = Math.Max(0, shapeBounds.Height - (2 * margin) - headerOffset);
var x = shapeBounds.X + margin;
var y = shapeBounds.Y + margin + headerOffset;

return new Rect(x, y, width, height);
}
else
{
double elementWidth = double.IsNaN(this.headerElement.Width) ? this.actualHeaderWidth :
this.headerElement.Width;
double headerWidth = double.IsNaN(elementWidth) ? this.headerElement.ActualWidth :
elementWidth;
headerWidth = Math.Round(headerWidth, 1);
var width = Math.Max(0, shapeBounds.Width - (2 * margin) - headerWidth);
var height = Math.Max(0, shapeBounds.Height - (2 * margin));
var x = shapeBounds.X + margin + headerWidth;
var y = shapeBounds.Y + margin;

var newBounds = new Rect(x, y, width, height);

return newBounds;
}
}

Rect Telerik.Windows.Diagrams.Core.IContainerShape.ContentBounds
{
get
{
return this.ContentBounds;
}
set
{
if
(!this.Diagram.ServiceLocator.GetService<Telerik.Windows.Diagrams.Core.IRotationService>().Is
Rotating)
{
```

```
this.ContentBounds = value;
}
}
}

protected virtual void OnHeaderPresenterSizeChanged(object sender, SizeChangedEventArgs e)
{
if (this.loaded)
{
var element = sender as FrameworkElement;
if (element != null && this.headerElement != null)
{
this.actualHeaderWidth = Math.Round(Math.Max(e.NewSize.Height + element.Margin.Top +
element.Margin.Bottom, this.headerElement.MinWidth));
}
else
{
this.actualHeaderWidth = e.NewSize.Height;
}
}
else if (this.headerElement != null)
{
this.actualHeaderWidth = Math.Max(this.headerElement.ActualWidth,
this.headerElement.MinWidth);
}
}

protected virtual void OnCustomContainerLoaded(object sender, RoutedEventArgs e)
{
if (this.templateApplied)
this.ContentBounds = this.GetNewContentBounds(this.Bounds);

this.loaded = true;
}

protected override void OnRotationAngleChanged(double newValue, double oldValue)
{
if (!this.isRollback)
{
this.isRollback = true;
this.RotationAngle = oldValue;
this.isRollback = false;
```

```
}
}

protected override void UpdateChildrenPositions(Point oldPosition, Point newPosition)
{
if (this.ShouldUpdateChildren)
base.UpdateChildrenPositions(oldPosition, newPosition);
}

protected override void OnChildBoundsChanged(Telerik.Windows.Diagrams.Core.IDiagramItem
diagramItem)
{
if (this.IsNotUserInteration())
base.OnChildBoundsChanged(diagramItem);
}

protected override void OnIsInEditModeChanged(bool oldIsInEditMode, bool isInEditMode)
{
base.OnIsInEditModeChanged(oldIsInEditMode, isInEditMode);
if (this.editPopup == null) return;

if (isInEditMode)
{
this.editPopup.IsOpen = true;
}
else
{
this.editPopup.IsOpen = false;
}
}

protected override Rect CalculateContentBounds(Rect shapeBounds)
{
return this.GetNewContentBounds(shapeBounds);
}

protected override Rect CalculateShapeBounds(Rect contentBounds)
{
if (this.headerElement == null) return contentBounds;

var shapeBounds = this.GetShapeBounds(contentBounds);
this.SetEditElementPosition(shapeBounds.Height);
```

```
return shapeBounds;
}

protected override void OnIsCollapsedChanged(bool newValue, bool oldValue)
{
if (this.Orientation == Orientation.Vertical)
{
base.OnIsCollapsedChanged(newValue, oldValue);
}
else
{
if (newValue)
{
var tmpMinHeight = this.MinHeight;
var tmpHeight = this.Height;

base.OnIsCollapsedChanged(newValue, oldValue);

this.MinHeight = tmpMinHeight;
this.Height = tmpHeight;

if (this.MinWidth > 0)
{
this.restoredMinWidth = this.MinWidth;
this.SetValue(MinWidthProperty,
MinWidthProperty.GetMetadata(typeof(FrameworkElement)).DefaultValue);
}

if (this.Width > 0)
{
this.restoredWidth = this.Width;
this.SetValue(WidthProperty,
WidthProperty.GetMetadata(typeof(FrameworkElement)).DefaultValue);
}
}
else
{
base.OnIsCollapsedChanged(newValue, oldValue);
this.Width = this.restoredWidth;
this.MinWidth = this.restoredMinWidth;
}
```

```
}
}

protected bool IsNotUserInteration()
{
return
!this.Diagram.ServiceLocator.GetService<Telerik.Windows.Diagrams.Core.IResizingService>().Is
Resizing &&
!this.Diagram.ServiceLocator.GetService<Telerik.Windows.Diagrams.Core.IDraggingService>().Is
Dragging &&
!this.Diagram.ServiceLocator.GetService<Telerik.Windows.Diagrams.Core.IRotationService>().Is
Rotating;
}

protected SwimlaneShapeBase GetSwimlane(Telerik.Windows.DragDrop.DragEventArgs e)
{
var swimlane = (e.Data as DataObject).GetData(typeof(SwimlaneShapeBase)) as
SwimlaneShapeBase;
if (swimlane == null)
{
var data = (e.Data as DataObject).GetData(typeof(DiagramDropInfo));
if (data != null)
{
var dropInfo = (DiagramDropInfo)data;
var items =
Telerik.Windows.Diagrams.Core.SerializationService.Default.DeserializeItems(dropInfo.Info, true);
return items.FirstOrDefault(i => i is SwimlaneShapeBase) as SwimlaneShapeBase;
}
}

return swimlane;
}

protected MainContainerShapeBase
GetMainContainer(Telerik.Windows.DragDrop.DragEventArgs e)
{
var mainContainer = (e.Data as DataObject).GetData(typeof(MainContainerShapeBase)) as
MainContainerShapeBase;
if (mainContainer == null)
{
var data = (e.Data as DataObject).GetData(typeof(DiagramDropInfo));
if (data != null)
```

```
{
var dropInfo = (DiagramDropInfo)data;
var items =
Telerik.Windows.Diagrams.Core.SerializationService.Default.DeserializeItems(dropInfo.Info, true);
return items.FirstOrDefault(i => i is MainContainerShapeBase) as MainContainerShapeBase;
}
}

return mainContainer;
}

protected Rect GetShapeBounds(Rect contentBounds)
{
if (this.headerElement == null) return contentBounds;

double margin = Telerik.Windows.Diagrams.Core.DiagramConstants.ContainerMargin;

if (this.Orientation == Orientation.Vertical)
{
double headerOffset = this.headerElement != null ? this.headerElement.ActualHeight +
this.headerElement.Margin.Top + this.headerElement.Margin.Bottom : 0;

var width = Math.Max(0, contentBounds.Width + (2 * margin));
var height = Math.Max(0, contentBounds.Height + (2 * margin) + headerOffset);
var x = contentBounds.X - margin;
var y = contentBounds.Y - margin - headerOffset;

return new Rect(x, y, width, height);
}
else
{
double elementWidth = double.IsNaN(this.headerElement.Width) ? actualHeaderWidth :
this.headerElement.Width;
double headerWidth = double.IsNaN(elementWidth) ? this.headerElement.ActualWidth :
elementWidth;
headerWidth = Math.Round(headerWidth, 1);
var width = contentBounds.Width + (2 * margin) + headerWidth;
var height = contentBounds.Height + (2 * margin);
var x = contentBounds.X - margin - headerWidth;
var y = contentBounds.Y - margin;

return new Rect(x, y, width, height);
```

```
}
}

private void SetEditElementPosition(double height)
{
}

private void OnEditPopupKeyDown(object sender, KeyEventArgs e)
{
if (e.Key == Key.Enter || e.Key == Key.Escape)
{
this.IsInEditMode = false;
}
}
}
}
```

CustomManipulationAdorner.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using Telerik.Windows.Controls.Diagrams.Primitives;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class CustomManipulationAdorner : ManipulationAdorner
{
public static readonly DependencyProperty AdditionalResizeHandlersProperty =
DependencyProperty.Register("AdditionalResizeHandlers",
typeof(ObservableCollection<ResizeHandler>), typeof(CustomManipulationAdorner), new
PropertyMetadata(null));

public static readonly DependencyProperty AdditionalHandlersVisibilityProperty =
DependencyProperty.Register("AdditionalHandlersVisibility", typeof(Visibility),
typeof(CustomManipulationAdorner), new PropertyMetadata(Visibility.Collapsed));
```

```
public ObservableCollection<ResizeHandler> AdditionalResizeHandlers
{
get { return (ObservableCollection<ResizeHandler>)GetValue(AdditionalResizeHandlersProperty);
}
set { SetValue(AdditionalResizeHandlersProperty, value); }
}

public Visibility AdditionalHandlersVisibility
{
get { return (Visibility)GetValue(AdditionalHandlersVisibilityProperty); }
set { SetValue(AdditionalHandlersVisibilityProperty, value); }
}

public CustomManipulationAdorner()
: base()
{
this.AdditionalResizeHandlers = new ObservableCollection<ResizeHandler>();
}

internal void UpdateAdditionalHandlers(List<SwimlaneShapeBase> containers, Point
boundsPosition, bool isVisible, Orientation orientation)
{
int i = 0;
if (isVisible && containers != null)
{
for (; i < containers.Count - 1; i++)
{
var bounds = containers[i].Bounds;
bounds.X -= boundsPosition.X;
bounds.Y -= boundsPosition.Y;
Point position = new Point(bounds.Left + (bounds.Width / 2) - 3, bounds.Bottom - 3);
if (orientation == System.Windows.Controls.Orientation.Horizontal)
position = new Point(bounds.Right - 3, bounds.Top + (bounds.Height / 2) - 3);
if (double.IsNaN(position.X) || double.IsNaN(position.Y) ||
double.IsInfinity(position.X) || double.IsInfinity(position.Y))
continue;

ResizeHandler handler = null;
if (i < this.AdditionalResizeHandlers.Count)
{
handler = this.AdditionalResizeHandlers[i];
}
```

```
else
{
handler = new ResizeHandler();
this.AdditionalResizeHandlers.Add(handler);
}
handler.X = position.X;
handler.Y = position.Y;
handler.TopShape = containers[i];
handler.BottomShape = containers[i + 1];
handler.Orientation = orientation;
handler.Cursor = orientation == Orientation.Vertical ? System.Windows.Input.Cursors.SizeNS :
System.Windows.Input.Cursors.SizeWE;
}
}

for (int j = this.AdditionalResizeHandlers.Count - 1; j >= i; j--)
{
this.AdditionalResizeHandlers.RemoveAt(j);
}
}
}



public class ResizeHandler : Control
{
private Point? startPosition;
private double topShapeStart;
private double start;
private double max;
private double min;

public ResizeHandler()
{
}

protected override void OnMouseLeftButtonDown(MouseButtonEventArgs e)
{
base.OnMouseLeftButtonDown(e);

this.TopShape.UpdateMinBounds();
this.BottomShape.UpdateMinBounds();
if (this.Orientation == System.Windows.Controls.Orientation.Vertical)
```

```
{
this.min = this.TopShape.MinBounds != Rect.Empty ? this.TopShape.MinBounds.Bottom :
this.TopShape.Bounds.Top + CustomResizingService.MinShapeHeight;
this.max = this.BottomShape.MinBounds != Rect.Empty ? this.BottomShape.MinBounds.Top :
this.BottomShape.Bounds.Bottom - CustomResizingService.MinShapeHeight;
this.topShapeStart = this.TopShape.Bounds.Bottom;
this.start = this.Y;
}
else
{
this.min = this.TopShape.MinBounds != Rect.Empty ? this.TopShape.MinBounds.Right :
this.TopShape.Bounds.Left + CustomResizingService.MinShapeWidth;
this.max = this.BottomShape.MinBounds != Rect.Empty ? this.BottomShape.MinBounds.Left :
this.BottomShape.Bounds.Right - CustomResizingService.MinShapeWidth;
this.topShapeStart = this.TopShape.Bounds.Right;
this.start = this.X;
}

this.CaptureMouse();
this.startPosition = e.GetPosition(null);
this.TopShape.IsInnerResize = true;
this.BottomShape.IsInnerResize = true;

e.Handled = true;
}

protected override void OnMouseLeftButtonUp(MouseButtonEventArgs e)
{
base.OnMouseLeftButtonUp(e);

this.ReleaseMouseCapture();
this.startPosition = null;
this.TopShape.IsInnerResize = false;
this.BottomShape.IsInnerResize = false;

e.Handled = true;
}

protected override void OnMouseMove(MouseEventArgs e)
{
base.OnMouseMove(e);
if (this.startPosition.HasValue)
```

```
{
var newPosition = e.GetPosition(null);
if (this.Orientation == System.Windows.Controls.Orientation.Vertical)
{
var offset = newPosition.Y - this.startPosition.Value.Y;
offset = Math.Max(offset, this.min - this.topShapeStart);
offset = Math.Min(offset, this.max - this.topShapeStart);

var topShapeBottom = this.topShapeStart + offset;

this.TopShape.Height = topShapeBottom - this.TopShape.Position.Y;

this.BottomShape.Height = this.BottomShape.Bounds.Bottom - topShapeBottom +
this.BottomShape.BorderThickness.Top;
this.BottomShape.Position = new Point(this.BottomShape.Position.X, topShapeBottom -
this.BottomShape.BorderThickness.Top);

this.Y = this.start + offset;
}
else
{
var offset = newPosition.X - this.startPosition.Value.X;
offset = Math.Max(offset, this.min - this.topShapeStart);
offset = Math.Min(offset, this.max - this.topShapeStart);

var topShapeEnd = this.topShapeStart + offset;

this.TopShape.Width = topShapeEnd - this.TopShape.Position.X;

this.BottomShape.Width = this.BottomShape.Bounds.Right - topShapeEnd +
this.BottomShape.BorderThickness.Left;
this.BottomShape.Position = new Point(topShapeEnd - this.BottomShape.BorderThickness.Left,
this.BottomShape.Position.Y);

this.X = this.start + offset;
}
}
}

public double X
{
get
```

```
{
return Canvas.GetLeft(this);
}
set
{
Canvas.SetLeft(this, value);
}
}

public double Y
{
get
{
return Canvas.GetTop(this);
}
set
{
Canvas.SetTop(this, value);
}
}

public SwimlaneShapeBase TopShape { get; set; }

public SwimlaneShapeBase BottomShape { get; set; }

public Orientation Orientation { get; set; }
}
}
```

CustomResizingService.cs

```
using System;
using System.Linq;
using System.Windows;
using Telerik.Windows.Controls;
using Telerik.Windows.Diagrams.Core;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class CustomResizingService : ResizingService
{
public static double MinShapeHeight = 12;
public static double MinShapeWidth = 12;
```

```
private MainContainerShapeBase mainContainer;
private CompositeAsyncStateCommand compositeCommand;
private bool isFirstChildResize = false;
private bool restrictResize = false;
private Rect startBounds;
private double min;
private double max;
//private bool isFirstResize;

public CustomResizingService(RadDiagram diagram)
: base(diagram)
{ }

public override void InitializeResize(System.Collections.Generic.IEnumerable<IDiagramItem>
newSelectedItems, double adornerAngle, Rect adornerBounds, ResizeDirection resizingDirection,
Point startPoint)
{
this.mainContainer = newSelectedItems.FirstOrDefault() as MainContainerShapeBase;
if (this.mainContainer != null)
this.mainContainer.UpdateMinBounds();

base.InitializeResize(newSelectedItems, adornerAngle, adornerBounds, resizingDirection,
startPoint);

if (this.mainContainer != null)
{
var firstChild = this.mainContainer.OrderedChildren.FirstOrDefault();
var lastChild = this.mainContainer.OrderedChildren.LastOrDefault();

if (firstChild != null && lastChild != null)
{
this.restrictResize = true;
this.startBounds = this.mainContainer.ContentBounds;

if (this.mainContainer.ChildrenPositioning == System.Windows.Controls.Orientation.Vertical)
{
this.isFirstChildResize = resizingDirection == ResizeDirection.NorthEastSouthWest ||
resizingDirection == ResizeDirection.NorthWestSouthEast;
this.min = firstChild.MinBoundsWithChildren != Rect.Empty ?
firstChild.MinBoundsWithChildren.Top : firstChild.Bounds.Bottom -
CustomResizingService.MinShapeHeight;
```

```
this.max = lastChild.MinBoundsWithChildren != Rect.Empty ?
lastChild.MinBoundsWithChildren.Bottom : lastChild.Bounds.Top +
CustomResizingService.MinShapeHeight;
}
else
{
this.isFirstChildResize = resizingDirection == ResizeDirection.SouthWestNorthEast ||
resizingDirection == ResizeDirection.NorthWestSouthEast;
this.min = firstChild.MinBoundsWithChildren != Rect.Empty ?
firstChild.MinBoundsWithChildren.Left : firstChild.Bounds.Right -
CustomResizingService.MinShapeWidth;
this.max = lastChild.MinBoundsWithChildren != Rect.Empty ?
lastChild.MinBoundsWithChildren.Right : lastChild.Bounds.Left +
CustomResizingService.MinShapeWidth;
}
this.CreateResizeCommand();
}
else
{
this.restrictResize = false;
}
}

//this.isFirstResize = true;
}

public override void Resize(Point newPoint)
{
//if (this.isFirstResize)
//    this.Graph.ServiceLocator.GetService<IUndoRedoService>();
//this.isFirstResize = false;

base.Resize(newPoint);

if (this.mainContainer != null)
{
var newBounds = this.mainContainer.GetNewContentBounds(this.mainContainer.Bounds);
if (this.mainContainer.ChildrenPositioning == System.Windows.Controls.Orientation.Vertical)
{
for (int i = 0; i < this.mainContainer.OrderedChildren.Count; i++)
{
var container = this.mainContainer.OrderedChildren[i];
```

```
if (!container.IsCollapsed)
container.Width = newBounds.Width;

if (i == 0)
{
container.Height = Math.Max(0, container.Bounds.Bottom - newBounds.Top);
container.Position = new Point(newBounds.X, newBounds.Y);
}
else
{
container.Position = new Point(newBounds.X, container.Position.Y);
}

if (i == this.mainContainer.OrderedChildren.Count - 1)
{
container.Height = Math.Max(0, newBounds.Bottom - container.Position.Y);
}
}
}
else
{
for (int i = 0; i < this.mainContainer.OrderedChildren.Count; i++)
{
var container = this.mainContainer.OrderedChildren[i];
if (!container.IsCollapsed)
container.Height = newBounds.Height;


if (i == 0)
{
container.Width = Math.Max(0, container.Bounds.Right - newBounds.Left);
container.Position = new Point(newBounds.X, newBounds.Y);
}
else
{
container.Position = new Point(container.Position.X, newBounds.Y);
}

if (i == this.mainContainer.OrderedChildren.Count - 1)
{
container.Width = Math.Max(0, newBounds.Right - container.Position.X);
}
```

```
}
}
}
}

public override void CompleteResize(Rect finalBounds, Point mousePosition)
{
var resizeCommand =
this.Graph.ServiceLocator.GetService<IUndoRedoService>().UndoStack.ElementAt(0) as
CompositeAsyncStateCommand;
if (resizeCommand != null)
{
resizeCommand.InsertCommand(0, new UndoableDelegateCommand("Set property",
new Action<object>((o) =>
{
this.mainContainer.ShouldUpdateChildren = false;
}),
new Action<object>((o) =>
{
this.mainContainer.ShouldUpdateChildren = true;
})), null);
resizeCommand.AddCommand(this.compositeCommand);
}

base.CompleteResize(finalBounds, mousePosition);

if (this.mainContainer != null)
{
var newStates = this.mainContainer.Children.OfType<IShape>().Select(s => new
ShapeInfo(s.Position, s.Bounds.ToSize(), s.RotationAngle)).OfType<object>();
if (this.compositeCommand != null)
this.compositeCommand.Complete(true, newStates);
}
}

protected override Point CalculateNewDelta(Point newPoint)
{
var resizeVector = base.CalculateNewDelta(newPoint);
if (this.mainContainer != null && this.restrictResize)
{
if (this.mainContainer.ChildrenPositioning == System.Windows.Controls.Orientation.Vertical)
{
```

```
if (this.isFirstChildResize)
{
if (this.startBounds.Top - resizeVector.Y > min)
resizeVector = new Point(resizeVector.X, Math.Min(0, this.startBounds.Top - min));

}
else
{
if (this.startBounds.Bottom + resizeVector.Y < max)
resizeVector = new Point(resizeVector.X, Math.Min(0, max - this.startBounds.Bottom));
}
}
else
{
if (this.isFirstChildResize)
{
if (this.startBounds.Left - resizeVector.X > min)
resizeVector = new Point(Math.Min(0, this.startBounds.Left - min), resizeVector.Y);

}
else
{
if (this.startBounds.Right + resizeVector.X < max)
resizeVector = new Point(Math.Min(0, max - this.startBounds.Right), resizeVector.Y);
}
}
}

return resizeVector;
}

protected override void UpdateContainers()
{
}

private void CreateResizeCommand()
{
var shapeChildren = this.mainContainer.Children.OfType<SwimlaneShapeBase>();
this.compositeCommand = new CompositeAsyncStateCommand("Resize horizontal containers");
compositeCommand.AddCommand(new UndoableDelegateCommand("Set property",
new Action<object>((o) =>
{
```

```
if (this.mainContainer != null)
this.mainContainer.ShouldUpdateChildren = true;
}),
new Action<object>((o) =>
{
if (this.mainContainer != null)
this.mainContainer.ShouldUpdateChildren = false;
})), null);
var commands = shapeChildren.Select(shape => new AsyncStateCommand("Resize horizontal
container",
state =>
{
shape.ShouldUpdateChildren = false;
var info = (ShapeInfo)state;
shape.Width = info.Size.Width;
shape.Height = info.Size.Height;
shape.Position = info.Position;
shape.ShouldUpdateChildren = true;
},
state =>
{
shape.ShouldUpdateChildren = false;
var info = (ShapeInfo)state;
shape.Width = info.Size.Width;
shape.Height = info.Size.Height;
shape.Position = info.Position;
shape.ShouldUpdateChildren = true;
}));
var states = shapeChildren.Select(s => new ShapeInfo(s.Position, s.Bounds.ToSize(),
s.RotationAngle));
for (int i = 0; i < commands.Count(); i++)
{
if (i < states.Count())
this.compositeCommand.AddCommand(commands.ElementAt(i), states.ElementAt(i));
else
this.compositeCommand.AddCommand(commands.ElementAt(i), null);
}
}
}
}
```

DataType.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public enum DataType
{
String,
Integer,
Double,
Stream,
DateTime,
Float
}
}
```

DataTypeExtension.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Markup;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
/// <summary>
/// DataType MarkupExtension.
/// </summary>
public class DataTypeExtension : MarkupExtension
{
/// <summary>
/// Initializes a new instance of the <see cref="DataTypeExtension"/> class.
/// </summary>
public DataTypeExtension()
{
}

public override object ProvideValue(IServiceProvider serviceProvider)
```

```
{
return Enum.GetValues(typeof(DataType));
}
}
}
```

MainContainerShapeBase.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Telerik.Windows.Controls;
using Telerik.Windows.Diagrams.Core;
using Telerik.Windows.DragDrop;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class MainContainerShapeBase : CustomContainerBase, IContainerShape
{
private static readonly double childrenMargin = 0;
private static readonly Size newItemSize = new Size(200, 110);

private readonly List<IConnection> collapsedItems;
private List<SwimlaneShapeBase> orderedChildren;
private bool isInternalResize;

public List<SwimlaneShapeBase> OrderedChildren
{
get
{
return this.orderedChildren;
}
}

public System.Windows.Controls.Orientation ChildrenPositioning { get; set; }

public MainContainerShapeBase()
{
this.ChildrenPositioning = System.Windows.Controls.Orientation.Vertical;
DragDropManager.AddDragEnterHandler(this, this.OnDragEnterManager);
```

```
this.collapsedItems = new List<IConnection>();
}

public void UpdateMinBounds()
{
foreach (var item in this.Children.OfType<SwimlaneShapeBase>())
{
item.UpdateMinBounds();
}

this.MinBounds = this.CalculateMinShapeBounds();
}

public override void OnApplyTemplate()
{
base.OnApplyTemplate();

this.UpdateChildContainers();
}

public void MoveTo(SwimlaneShapeBase dragObject, int endPosition, bool isUndoable = true)
{
bool shouldAdd = !this.Items.Contains(dragObject);
bool addToDiagram = !this.Diagram.Items.Contains(dragObject);

var diagramPosition = dragObject.Position;
var startPosition = shouldAdd ? this.Items.Count : dragObject.ContainerPosition;
endPosition = startPosition <= endPosition ? endPosition : endPosition + 1;

var doAction = new Action<object>((o) =>
{
this.MoveContainer(startPosition, endPosition, dragObject);
if (shouldAdd)
this.AddItem(dragObject);
this.UpdateChildContainers();
});

if (isUndoable)
{
var command = new UndoableDelegateCommand("Move container",
doAction,
new Action<object>((o) =>
```

```
{
this.MoveContainer(endPosition, startPosition, dragObject);
if (shouldAdd)
{
this.RemoveItem(dragObject);
dragObject.Position = diagramPosition;
if (addToDiagram)
this.Diagram.Items.Remove(dragObject);
}
this.UpdateChildContainers();
}));
this.Diagram.ServiceLocator.GetService<IUndoRedoService>().ExecuteCommand(command);
}
else
{
doAction.Invoke(null);
}
}

public void UpdateChildContainers(bool resizeChildren = false)
{
var bounds = this.ContentBounds;
this.orderedChildren = this.Children.OfType<SwimlaneShapeBase>().OrderBy(c =>
c.ContainerPosition).ToList();
if (!this.loaded || !this.templateApplied) return;

if (orderedChildren.Count > 0)
{
if (this.ChildrenPositioning == System.Windows.Controls.Orientation.Vertical)
{
double childWidth = bounds.Width;
if (!resizeChildren)
childWidth = Math.Max(bounds.Width, this.OrderedChildren.Max(c => c.ContentBounds.Width));
for (int i = 0; i < this.orderedChildren.Count; i++)
{
var container = this.orderedChildren[i];
if (i == 0)
container.Position = new Point(bounds.X, bounds.Y);
else
container.Position = new Point(bounds.X, this.orderedChildren[i - 1].Bounds.Bottom - 1 +
childrenMargin);
```

```
if (double.IsNaN(container.Height) || container.Height == 0)
container.Height = newItemSize.Height;

container.Width = childWidth;

if (i == this.orderedChildren.Count - 1 && container.Bounds.Bottom != bounds.Bottom)
{
bool update = true;
if (resizeChildren)
{
var minBottom = container.MinBounds != Rect.Empty ? container.MinBounds.Bottom :
container.Bounds.Top + CustomResizingService.MinShapeHeight;
if (minBottom > bounds.Bottom)
{
container.Height = minBottom - container.Y;
}
else
{
container.Height = bounds.Bottom - container.Y;
update = false;
}
}

if (update)
{
this.isInternalResize = true;
this.ContentBounds = new Rect(this.ContentBounds.TopLeft(), new Size(childWidth,
container.Bounds.Bottom - this.ContentBounds.Y));
}
}
}
}
else
{
double childHeight = bounds.Height;
if (!resizeChildren)
childHeight = Math.Max(bounds.Height, this.OrderedChildren.Max(c => c.ContentBounds.Height));
for (int i = 0; i < this.orderedChildren.Count; i++)
{
var container = this.orderedChildren[i];
if (i == 0)
container.Position = new Point(bounds.X, bounds.Y);
```

```
else
container.Position = new Point(this.orderedChildren[i - 1].Bounds.Right - 1 + childrenMargin,
bounds.Y);

if (double.IsNaN(container.Width) || container.Width == 0)
container.Width = newItemSize.Width;

container.Height = childHeight;

if (i == this.orderedChildren.Count - 1 && container.Bounds.Right != bounds.Right)
{
bool update = true;
if (resizeChildren)
{
var minRight = container.MinBounds != Rect.Empty ? container.MinBounds.Right :
container.Bounds.Left + CustomResizingService.MinShapeWidth;
if (minRight > bounds.Right)
{
container.Width = minRight - container.X;
}
else
{
container.Width = bounds.Right - container.X;
update = false;
}
}

if (update)
{
this.isInternalResize = true;
this.ContentBounds = new Rect(this.ContentBounds.TopLeft(), new Size(container.Bounds.Right -
bounds.X, childHeight));
}
}
}
}
}
}

void IContainerShape.AddItem(object item, Point? position)
{
var swimLaneItem = item as SwimlaneShapeBase;
```

```
if (this.loaded && swimLaneItem != null)
{
var endPos = swimLaneItem.ContainerPosition;
swimLaneItem.ContainerPosition = this.Items.Count;
this.MoveTo(swimLaneItem, endPos - 1, false);
this.TryToSelect();
}
else if (swimLaneItem != null || item is IConnection)
{
base.AddItem(item);
this.TryToSelect();
}
}

void IContainerShape.AddItems(IEnumerable<object> items)
{
if (items.Any(c => c is SwimlaneShapeBase))
{
this.AddItems(items);
this.UpdateChildContainers();
this.TryToSelect();
base.OnDragLeave(new DragItemsEventArgs());
}
}

internal void SwapChildren(SwimlaneShapeBase first, SwimlaneShapeBase second)
{
int firstPosition = first.ContainerPosition;
int secondPosition = second.ContainerPosition;
var command = new UndoableDelegateCommand("Swap children",
new Action<object>((o) =>
{
first.ContainerPosition = secondPosition;
second.ContainerPosition = firstPosition;
this.UpdateChildContainers();
}),
new Action<object>((o) =>
{
first.ContainerPosition = firstPosition;
second.ContainerPosition = secondPosition;
this.UpdateChildContainers();
}));
```

```
this.Diagram.ServiceLocator.GetService<IUndoRedoService>().ExecuteCommand(command);
}

protected override void OnSizeChanged(Size newSize, Size oldSize)
{
base.OnSizeChanged(newSize, oldSize);
if (this.loaded && !this.isInternalResize && this.Diagram != null &&
!this.Diagram.ServiceLocator.GetService<IResizingService>().IsResizing)
{
this.UpdateChildContainers(true);
}
this.isInternalResize = false;
}

protected override void OnHeaderPresenterSizeChanged(object sender, SizeChangedEventArgs
e)
{
base.OnHeaderPresenterSizeChanged(sender, e);
if (this.loaded)
{
this.ShouldUpdateChildren = false;
var newX = this.ContentBounds.X - DiagramConstants.ContainerMargin - this.actualHeaderWidth;
var newWidth = this.Bounds.Right - newX;
var widthDelta = newWidth - this.Width;

this.Position = new Point(newX, this.Position.Y);
this.Width = Math.Max(0, newWidth);
if (this.IsCollapsed && e.PreviousSize.Height != 0)
{
if (widthDelta.IsNanOrInfinity() || widthDelta == 0)
widthDelta = e.NewSize.Height - e.PreviousSize.Height;

this.SetValue(WidthProperty,
WidthProperty.GetMetadata(typeof(FrameworkElement)).DefaultValue);
this.restoredWidth = Math.Max(0, this.restoredWidth + widthDelta);
}
this.ShouldUpdateChildren = true;
}
}

protected override void OnDrop(DragItemsEventArgs args)
```

```
{
}

protected override void OnDragEnter(DragItemsEventArgs args)
{
if (args.Items.Count() > 0 && args.Items.Any(c => c is SwimlaneShapeBase))
base.OnDragEnter(args);
}

protected override void OnManagerDrop(object sender,
Telerik.Windows.DragDrop.DragEventArgs e)
{
SwimlaneShapeBase swimlane = this.GetSwimlane(e);
if (swimlane != null)
{
VisualStateManager.GoToState(this, "DropNormal", false);
var tmpCollapsed = this.IsCollapsed;
if (tmpCollapsed)
this.IsCollapsed = false;

if (swimlane != null)
this.MoveTo(swimlane, this.Items.Count);

this.TryToSelect();

this.IsCollapsed = tmpCollapsed;

e.Handled = true;
}
}

protected override Rect CalculateMinShapeBounds()
{
var minBounds = Rect.Empty;
foreach (var item in this.Children.OfType<SwimlaneShapeBase>())
{
minBounds.Union(item.MinBounds);
}
if (minBounds != Rect.Empty)
{
return this.GetShapeBounds(minBounds);
}
```

```
return minBounds;
}

protected override void OnItemsCollectionChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
{
base.OnItemsCollectionChanged(sender, e);
this.IsDropEnabled = this.Items.Count == 0;
if (e.Action == System.Collections.Specialized.NotifyCollectionChangedAction.Add)
{
var container = e.NewItems[0] as SwimlaneShapeBase;
if (container != null)
{
container.IsResizingEnabled = false;
}
}
else if (e.Action == System.Collections.Specialized.NotifyCollectionChangedAction.Remove)
{
var container = e.OldItems[0] as FrameworkElement;
if (container != null)
{
container.ClearValue(RadDiagramShape.IsResizingEnabledProperty);
}
}

this.orderedChildren = this.Children.OfType<SwimlaneShapeBase>().OrderBy(c =>
c.ContainerPosition).ToList();
//this.ContentBounds = this.CalculateContentBounds(this.Bounds);
this.UpdateChildContainers();
}

protected override void OnCustomContainerLoaded(object sender, RoutedEventArgs e)
{
base.OnCustomContainerLoaded(sender, e);
this.UpdateChildContainers();
}

private void TryToSelect()
{
var diagram = this.Diagram as RadDiagram;
if (diagram != null)
```

```
{
diagram.DeselectAll();
diagram.SelectedItem = this;
}
}

private void MoveContainer(int startPosition, int endPosition, SwimlaneShapeBase dragObject)
{
if (startPosition <= endPosition)
{
for (int i = 0; i < this.OrderedChildren.Count; i++)
{
var item = this.OrderedChildren[i];
if (item.ContainerPosition <= startPosition)
continue;
if (item.ContainerPosition > endPosition)
break;

item.ContainerPosition--;
}

dragObject.ContainerPosition = endPosition;
}
else
{
for (int i = 0; i < this.OrderedChildren.Count; i++)
{
var item = this.OrderedChildren[i];
if (item.ContainerPosition < endPosition)
continue;
if (item.ContainerPosition >= startPosition)
break;

item.ContainerPosition++;
}

dragObject.ContainerPosition = endPosition;
}
}

private void OnDragEnterManager(object sender, Telerik.Windows.DragDrop.DragEventArgs e)
{
```

```
if (this.GetSwimlane(e) != null)
base.OnDragEnter(new DragItemsEventArgs());
}
}
}
```

RowModel.cs

```
using Telerik.Windows.Controls.Diagrams.Extensions.ViewModels;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class RowModel : NodeViewModelBase
{
private string columnName;
private DataType dataType;

public DataType DataType
{
get
{
return this.dataType;
}
set
{
if (this.dataType != value)
{
this.dataType = value;
this.OnPropertyChanged("DataType");
}
}
}

public string ColumnName
{
get
{
return this.columnName;
}
set
{
if (this.columnName != value)
{
```

```
this.columnName = value;
this.OnPropertyChanged("ColumnName");
}
}
}
}
}
```

RowShape.cs

```
using System.Windows;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows.Diagrams.Core;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class RowShape : RadDiagramShapeBase
{
private readonly string connectionToolName = "Connection Tool";
private bool isButtonDown;
private Point lastDownPosition;

public RowShape()
: base()
{
}

public int ContainerPosition { get; set; }

protected override void OnMouseLeave(System.Windows.Input.MouseEventArgs e)
{
base.OnMouseLeave(e);
this.isButtonDown = false;
}

protected override void OnMouseLeftButtonDown(System.Windows.Input.MouseButtonEventArgs
e)
{
base.OnMouseLeftButtonDown(e);
this.lastDownPosition = e.GetPosition(this);
this.isButtonDown = true;
}
```

```
protected override void OnMouseLeftButtonUp(System.Windows.Input.MouseButtonEventArgs e)
{
base.OnMouseLeftButtonUp(e);
this.isButtonDown = false;
}

protected override void OnMouseMove(System.Windows.Input.MouseEventArgs e)
{
base.OnMouseMove(e);
if (this.isButtonDown && !this.IsInEditMode &&
!this.connectionToolName.Equals(this.Diagram.ServiceLocator.GetService<IToolService>().Active
Tool.Name))
{
if (!this.lastDownPosition.AroundPoint(e.GetPosition(this), 2))
{
var diagram = this.Diagram as RadDiagram;
if (diagram != null)
diagram.SelectedItem = diagram.ContainerGenerator.ItemFromContainer(this);
this.isButtonDown = false;
var toolService = this.Diagram.ServiceLocator.GetService<IToolService>();
toolService.ActivateTool(this.connectionToolName);
toolService.MouseDown(new PointerArgs());
}
}
else
{
this.lastDownPosition = new Point(-100, -100);
this.isButtonDown = false;
}
}
}
}
```

ShapeStyleSelector.cs

```
using System.Windows;
using System.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class ShapeStyleSelector : StyleSelector
{
```

```
public Style RowStyle { get; set; }
public Style TableStyle { get; set; }

public override System.Windows.Style SelectStyle(object item,
System.Windows.DependencyObject container)
{
if (item is RowModel)
return this.RowStyle;

return this.TableStyle;
}
}
}
```

sqlDiagram.cs

```
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Diagrams;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class SqlDiagram : RadDiagram
{
protected override Telerik.Windows.Diagrams.Core.IContainerShape
GetShapeContainerForItemOverride(Telerik.Windows.Diagrams.Core.IContainerItem item)
{
return new TableShape();
}

protected override Telerik.Windows.Diagrams.Core.IShape
GetShapeContainerForItemOverride(object item)
{
return new RowShape();
}

protected override bool IsItemItsOwnShapeContainerOverride(object item)
{
return item is RadDiagramShapeBase;
}

public override void Paste()
{
var selectedContainer = this.ContainerGenerator.ContainerFromItem(this.SelectedItem) as
```

```
TableShape;
if (selectedContainer != null)
base.Paste();
}
}
}
```

SwimlaneCommands.cs

```
using System.Windows.Input;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public static class SwimlaneCommands
{
private static RoutedUICommand addCommand;
private static RoutedUICommand removeCommand;
public static RoutedUICommand RemoveCommand
{
get
{
if (removeCommand == null)
removeCommand = new RoutedUICommand("Remove Command", "RemoveCommand",
typeof(SwimlaneCommands));
return removeCommand;
}
}

public static RoutedUICommand AddCommand
{
get
{
if (addCommand == null)
addCommand = new RoutedUICommand("Add Command", "AddCommand",
typeof(SwimlaneCommands));
return addCommand;
}
}
}
}
```

SwimlaneDiagram.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media.Imaging;
using sys = System.Windows.Controls;
using selection = System.Windows.Controls;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows.Diagrams.Core;
using Telerik.Windows.DragDrop;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class SwimlaneDiagram : RadDiagram
{
private CustomManipulationAdorner manipulationAdorner;
private FrameworkElement horizontalDragVisual;
private FrameworkElement verticalDragVisual;
private DiagramSurface itemHost;

public SwimlaneDiagram()
: base()
{
DragDropManager.AddDropHandler(this, OnDrop);
this.ItemsChanging += OnSwimlaneDiagramItemsChanging;
}

public override void OnApplyTemplate()
{
base.OnApplyTemplate();

this.manipulationAdorner = this.GetTemplateChild("ManipulationAdorner") as
CustomManipulationAdorner;
this.horizontalDragVisual = this.GetTemplateChild("PART_horizontalDragOver") as
FrameworkElement;
this.verticalDragVisual = this.GetTemplateChild("PART_verticalDragOver") as FrameworkElement;
this.itemHost = this.GetTemplateChild("ItemsHost") as DiagramSurface;

this.SelectionChanged += OnSelectionChanged;
}
```

```
protected override void OnDeleteCommandExecutedOverride(object sender,
ExecutedRoutedEventArgs e)
{
var compositeRemoveCommand = new CompositeCommand("Remove Connections");
foreach (var item in this.SelectedItems)
{
var container = this.ContainerGenerator.ContainerFromItem(item) as IShape;
if (container != null)
{
foreach (var connection in this.GetConnectionsForShape(container).ToList())
{
compositeRemoveCommand.AddCommand(new UndoableDelegateCommand("Remove
Connection",
new Action<object>((o) => this.RemoveConnection(connection)),
new Action<object>((o) => this.AddConnection(connection))));
}
}
}

base.OnDeleteCommandExecutedOverride(sender, e);

if (compositeRemoveCommand.Commands.Count() > 0)
{
compositeRemoveCommand.Execute();
((this.UndoRedoService.UndoStack.FirstOrDefault() as CompositeCommand).Commands as
IList<Telerik.Windows.Diagrams.Core.ICommand>).Add(compositeRemoveCommand);
}
}

protected override bool PublishDiagramEvent(DiagramEvent diagramEvent, object args)
{
if (diagramEvent == DiagramEvent.SelectionBoundsChanged)
{
var mainContainer = this.SelectedItem as MainContainerShapeBase;
UpdateHandlers(mainContainer);
}
return base.PublishDiagramEvent(diagramEvent, args);
}

internal WriteableBitmap CreateImage(Rect bounds)
{
```

```
return BitmapUtils.CreateWriteableBitmap(this.itemHost as UIElement, bounds, bounds.ToSize(),
null, new Thickness());
}

internal void ShowDragOverVisual(SwimlaneShapeBase shape)
{
if (shape.ParentMainContainer == null) return;

var bounds = shape.Bounds;
if (shape.ParentMainContainer.ChildrenPositioning == sys.Orientation.Vertical)
{
if (this.horizontalDragVisual == null) return;

this.horizontalDragVisual.Visibility = System.Windows.Visibility.Visible;
Canvas.SetLeft(this.horizontalDragVisual, bounds.X);
Canvas.SetTop(this.horizontalDragVisual, bounds.Bottom);
this.horizontalDragVisual.Width = bounds.Width;
}
else
{
if (this.verticalDragVisual == null) return;

this.verticalDragVisual.Visibility = System.Windows.Visibility.Visible;
Canvas.SetLeft(this.verticalDragVisual, bounds.Right);
Canvas.SetTop(this.verticalDragVisual, bounds.Y);
this.verticalDragVisual.Height = bounds.Height;
}
}

internal void HideDragOverVisual()
{
if (this.horizontalDragVisual != null)
this.horizontalDragVisual.Visibility = System.Windows.Visibility.Collapsed;
if (this.verticalDragVisual != null)
this.verticalDragVisual.Visibility = System.Windows.Visibility.Collapsed;
}

private void OnSelectionChanged(object sender, selection.SelectionChangedEventArgs e)
{
var mainContainer = this.SelectedItem as MainContainerShapeBase;
if (e.AddedItems.Count > 0 && this.manipulationAdorner != null && mainContainer != null)
{
```

```
this.manipulationAdorner.AdditionalHandlersVisibility = System.Windows.Visibility.Visible;
this.UpdateHandlers(mainContainer);
}
else
{
this.manipulationAdorner.AdditionalHandlersVisibility = System.Windows.Visibility.Collapsed;
}
}

private void OnDrop(object sender, Telerik.Windows.DragDrop.DragEventArgs e)
{
var swimlane = (e.Data as DataObject).GetData(typeof(SwimlaneShapeBase)) as
SwimlaneShapeBase;
if (swimlane != null)
{
var transformedPoint = this.GetTransformedPoint(e.GetPosition(this));
var newPosition = transformedPoint.Substract(swimlane.DragStartOffset);
var oldPosition = swimlane.Position;
var mainParent = swimlane.ParentContainer as MainContainerShapeBase;
if (mainParent != null)
{
var command = new UndoableDelegateCommand("Remove container from main",
new Action<object>((o) =>
{
swimlane.Position = newPosition;
mainParent.Items.Remove(swimlane);
mainParent.UpdateChildContainers();
}),
new Action<object>((o) =>
{
swimlane.Position = oldPosition;
mainParent.Items.Add(swimlane);
mainParent.UpdateChildContainers();
}));

this.UndoRedoService.ExecuteCommand(command);
}
else
{
var command = new UndoableDelegateCommand("Remove container from main",
new Action<object>((o) =>
{
```

```
swimlane.Position = newPosition;
}),
new Action<object>((o) =>
{
swimlane.Position = oldPosition;
}));

this.UndoRedoService.ExecuteCommand(command);
}
}
}

private void OnSwimlaneDiagramItemsChanging(object sender, DiagramItemsChangingEventArgs
e)
{
if (e.Action == System.Collections.Specialized.NotifyCollectionChangedAction.Remove)
{
var container = e.OldItems.ElementAt(0) as SwimlaneShapeBase;
if (container != null && container.ParentMainContainer != null)
{
foreach (var item in container.ParentMainContainer.OrderedChildren.Where(c =>
c.ContainerPosition > container.ContainerPosition))
{
item.ContainerPosition--;
}
}
}
}

private void UpdateHandlers(MainContainerShapeBase mainContainer)
{
if (mainContainer != null)
{
var offset = this.ServiceLocator.GetService<IAdornerService>().InflatedAdornerBounds.TopLeft();
if (this.manipulationAdorner != null && !offset.X.IsNanOrInfinity() && !offset.Y.IsNanOrInfinity())
{
this.manipulationAdorner.UpdateAdditionalHandlers(mainContainer.OrderedChildren, offset,
!mainContainer.IsCollapsed, mainContainer.ChildrenPositioning);
}
}
}
}
```

```
}
```

SwimlaneShapeBase.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows.Diagrams.Core;
using Telerik.Windows.DragDrop;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class SwimlaneShapeBase : CustomContainerBase, IContainerShape
{
private SwimlaneDiagram swimlaneDiagram;
private bool canDrag;
private bool keepBackground;

public static readonly DependencyProperty ContainerPositionProperty =
DependencyProperty.Register("ContainerPosition", typeof(int), typeof(SwimlaneShapeBase), new
PropertyMetadata(-1));

public SwimlaneShapeBase()
{
this.MinBoundsWithChildren = Rect.Empty;

DragDropManager.AddDragLeaveHandler(this, this.OnDragLeaveManager);
DragDropManager.AddDragEnterHandler(this, this.OnDragEnterManager);
DragDropManager.AddDragInitializeHandler(this, OnDragInit);
DragDropManager.AddDragDropCompletedHandler(this, OnDragComplete);
}

public Rect MinBoundsWithChildren { get; set; }
public bool IsInnerResize { get; set; }
public int ContainerPosition
{
get { return (int)GetValue(ContainerPositionProperty); }
set { SetValue(ContainerPositionProperty, value); }
}
```

```
public Point DragStartOffset { get; private set; }
internal MainContainerShapeBase ParentMainContainer
{
get
{
return this.ParentContainer as MainContainerShapeBase;
}
}

public override void OnApplyTemplate()
{
if (this.headerElement != null)
{
this.headerElement.MouseEnter -= OnHeaderElementMouseEnter;
this.headerElement.MouseLeave -= OnHeaderElementMouseLeave;
}

base.OnApplyTemplate();

if (this.headerElement != null)
{
this.headerElement.MouseEnter += OnHeaderElementMouseEnter;
this.headerElement.MouseLeave += OnHeaderElementMouseLeave;
}
}

void IContainerShape.AddItem(object item, Point? position)
{
if (item as SwimlaneShapeBase == null)
this.AddItem(item, position);
}

void IContainerShape.AddItems(IEnumerable<object> items)
{
if (!items.Any(c => c is SwimlaneShapeBase || c is MainContainerShapeBase))
this.AddItems(items);
}

public override SerializationInfo Serialize()
{
bool[] props = new bool[2] { this.IsResizingEnabled, this.IsDraggingEnabled };
```

```
bool[] restore = new bool[2] { false, false };
if (this.ReadLocalValue(SwimlaneShapeBase.IsResizingEnabledProperty) !=
DependencyProperty.UnsetValue)
{
restore[0] = true;
this.ClearValue(SwimlaneShapeBase.IsResizingEnabledProperty);
}
if (this.ReadLocalValue(SwimlaneShapeBase.IsDraggingEnabledProperty) !=
DependencyProperty.UnsetValue)
{
restore[1] = true;
this.ClearValue(SwimlaneShapeBase.IsDraggingEnabledProperty);
}

var result = base.Serialize();
result["ContainerPosition"] = this.ContainerPosition.ToString();

if (restore[0])
this.IsResizingEnabled = props[0];

if (restore[1])
this.IsDraggingEnabled = props[1];

return result;
}

public override void Deserialize(SerializationInfo info)
{
base.Deserialize(info);
if (info["ContainerPosition"] != null)
this.ContainerPosition = int.Parse(info["ContainerPosition"].ToString());
}

public void UpdateMinBounds()
{
this.MinBounds = this.GetMinBounds();
}

protected override void Initialize(IGraphServiceLocator serviceLocator, IGraphInternal graph)
{
base.Initialize(serviceLocator, graph);
this.swimlaneDiagram = this.Diagram as SwimlaneDiagram;
```

```
}

protected override void UpdateChildrenPositions(Point oldPosition, Point newPosition)
{
if (!this.IsInnerResize)
base.UpdateChildrenPositions(oldPosition, newPosition);
}

protected override void OnChildBoundsChanged(IDiagramItem diagramItem)
{
}

protected override Rect CalculateMinShapeBounds()
{
return this.GetMinBounds();
}

protected override void OnManagerDrop(object sender,
Telerik.Windows.DragDrop.DragEventArgs e)
{
this.swimlaneDiagram.HideDragOverVisual();

SwimlaneShapeBase swimlane = this.GetSwimlane(e);
if (swimlane != null && this != swimlane)
{
if (this.ParentMainContainer == null) return;

var transformedPoint = (this.Diagram as
RadDiagram).GetTransformedPoint(e.GetPosition(this.Diagram as UIElement));

this.ParentMainContainer.MoveTo(swimlane, this.ContainerPosition);

if (swimlane.IsPointOverHeaderElement(transformedPoint))
swimlane.keepBackground = true;

e.Handled = true;
}
else if (this.GetMainContainer(e) == null)
{
base.OnManagerDrop(sender, e);
}
}
```

```
protected override void OnDragEnter(DragItemsEventArgs args)
{
if (args.Items.Count() > 0 && !args.Items.Any(c => c is CustomContainerBase))
base.OnDragEnter(args);
}

protected override void OnMouseLeftButtonDown(System.Windows.Input.MouseButtonEventArgs
e)
{
if (!this.IsPointOverHeaderElement((this.Diagram as
RadDiagram).GetTransformedPoint(e.GetPosition(this.Diagram as UIElement))))
{
if (this.ParentMainContainer == null)
this.IsDraggingEnabled = true;
else
this.ClearValue(SwimlaneShapeBase.IsDraggingEnabledProperty);

base.OnMouseLeftButtonDown(e);
}
else
{
this.ClearValue(SwimlaneShapeBase.IsDraggingEnabledProperty);
}
}

private void OnDragEnterManager(object sender, Telerik.Windows.DragDrop.DragEventArgs e)
{
if (this.GetSwimlane(e) != null)
{
this.swimlaneDiagram.ShowDragOverVisual(this);
}
else if (this.GetMainContainer(e) == null)
{
base.OnDragEnter(new DragItemsEventArgs());
}
}

private void OnDragLeaveManager(object sender, Telerik.Windows.DragDrop.DragEventArgs e)
{
if (this.GetSwimlane(e) != null)
{
```

```
this.swimlaneDiagram.HideDragOverVisual();
}
else if (this.GetMainContainer(e) == null)
{
base.OnDragLeave(new DragItemsEventArgs());
}
}

private void OnHeaderElementMouseLeave(object sender, MouseEventArgs e)
{
this.OnHeaderElementLeave();
}

private void OnHeaderElementMouseEnter(object sender, MouseEventArgs e)
{
this.OnHeaderElementEnter();
}

public bool IsPointOverHeaderElement(Point transformedPoint)
{
if (this.headerElement == null) return false;

return new Rect(this.Position, new Size(this.headerElement.ActualWidth,
this.headerElement.ActualHeight)).Contains(transformedPoint);
}

private void OnHeaderElementLeave()
{
this.canDrag = false;
VisualStateManager.GoToState(this, "NormalHeader", false);
}

private void OnHeaderElementEnter()
{
this.canDrag = true;

VisualStateManager.GoToState(this, "MouseOverHeader", false);
}

private Rect GetShapeBounds(Rect contentBounds)
{
if (this.headerElement == null) return contentBounds;
```

```
if (this.Orientation == System.Windows.Controls.Orientation.Horizontal)
{
var width = contentBounds.Width + this.headerElement.ActualWidth;
var height = contentBounds.Height;
var x = contentBounds.X - this.headerElement.ActualWidth;
var y = contentBounds.Y;

return new Rect(x, y, width, height);
}
else
{
var width = contentBounds.Width;
var height = contentBounds.Height + this.headerElement.ActualHeight;
var x = contentBounds.X;
var y = contentBounds.Y - this.headerElement.ActualHeight;

return new Rect(x, y, width, height);
}
}

private void OnDragInit(object sender, DragInitializeEventArgs e)
{
if (this.canDrag)
{
e.Data = new DataObject(typeof(SwimlaneShapeBase), this);
this.Diagram.ServiceLocator.GetService<IToolService>().IsMouseDown = false;
this.DragStartOffset = e.RelativeStartPoint;

var newBounds = new Rect(this.Position, new Size(this.Bounds.Width, this.Height + 1));
var bm = this.swimlaneDiagram.CreateImage(newBounds);

var draggingImage = new System.Windows.Controls.Image
{
Source = bm,
Width = newBounds.Width,
Height = newBounds.Height,
};

e.DragVisual = new Border() { Child = draggingImage, Opacity = 0.5 };

e.AllowedEffects = DragDropEffects.All;
```

```
e.Handled = true;
}
else
{
e.Cancel = true;
}
}

private void OnDragComplete(object sender, DragDropCompletedEventArgs e)
{
if (!this.keepBackground)
this.OnHeaderElementLeave();

this.swimlaneDiagram.HideDragOverVisual();
this.keepBackground = false;
}

private Rect GetMinBounds()
{
var childrenBounds = this.GetChildrenBounds();

if (childrenBounds != Rect.Empty)
{
childrenBounds = this.GetShapeBounds(childrenBounds);
this.MinBoundsWithChildren = childrenBounds;
childrenBounds.Intersect(this.Bounds);
}
else
{
this.MinBoundsWithChildren = childrenBounds;
}

return childrenBounds;
}
}
}
```

TableAdditionalContent.cs

```
using System;
using System.Linq;
using System.Windows.Input;
using Telerik.Windows.Controls.Diagrams.Extensions.ViewModels;
```

```
using Telerik.Windows.Diagrams.Core;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class TableAdditionalContent : AdditionalContent
{
protected override void OnContextItemChanged(object newValue, object oldValue)
{
if (newValue is TableModel)
{
this.Visibility = System.Windows.Visibility.Visible;
if (this.addRemove != null)
this.addRemove.Visibility = System.Windows.Visibility.Visible;
if (this.settingsPane != null)
{
this.settingsPane.Opacity = 0;
this.settingsPane.IsHitTestVisible = false;
}
}
else
this.Visibility = System.Windows.Visibility.Collapsed;
}

protected override void OnAdd()
{
var model = this.ContextItem as TableModel;
if (model != null)
{
var itemToAdd = new RowModel() { ColumnName = "NewRow", DataType = DataType.String };
var addCommand = new UndoableDelegateCommand("Add new row", new Action<object>((o) =>
this.AddNewRow(model, itemToAdd)), new Action<object>((o) => this.RemoveRow(model,
itemToAdd)));
this.Diagram.UndoRedoService.ExecuteCommand(addCommand);
}
}

protected override void OnCanRemove(CanExecuteRoutedEventArgs e)
{
var model = this.ContextItem as TableModel;
if (model != null && model.InternalItems.Count > 0)
{
e.CanExecute = true;
```

```
}
}

protected override void OnRemove()
{
var model = this.ContextItem as TableModel;
if (model != null && model.InternalItems.Count > 0)
{
var itemToRemove = model.InternalItems.LastOrDefault();
var removeCommand = new UndoableDelegateCommand("Add new row", new Action<object>((o)
=> this.RemoveRow(model, itemToRemove)), new Action<object>((o) => this.AddNewRow(model,
itemToRemove)));
this.Diagram.UndoRedoService.ExecuteCommand(removeCommand);
}
}

private void AddNewRow(TableModel model, NodeViewModelBase itemToRemove)
{
if (model.IsCollapsed)
model.IsCollapsed = false;
if (itemToRemove == null)
model.AddItem(new RowModel() { ColumnName = "NewRow", DataType = DataType.String });
else
model.AddItem(itemToRemove);
}

private void RemoveRow(TableModel model, NodeViewModelBase itemToRemove)
{
var dc = this.DataContext as TablesGraphSource;
if (dc != null && model != null && itemToRemove != null)
{
if (model.IsCollapsed)
model.IsCollapsed = false;
dc.RemoveItem(itemToRemove);
}

CommandManager.InvalidateRequerySuggested();
}
}
}
```

TableGraphSource.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Telerik.Windows.Controls.Diagrams.Extensions.ViewModels;
using Telerik.Windows.Diagrams.Core;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class TablesGraphSource : SerializableGraphSourceBase<NodeViewModelBase,
LinkViewModelBase<NodeViewModelBase>>
{
public TablesGraphSource()
{
}

public override string GetNodeUniqueId(NodeViewModelBase node)
{
if (node != null)
return node.GetHashCode().ToString();

return string.Empty;
}

public override void SerializeNode(NodeViewModelBase node,
Telerik.Windows.Diagrams.Core.SerializationInfo info)
{
base.SerializeNode(node, info);

var row = node as RowModel;
if (row != null)
{
info["ColumnName"] = row.ColumnName.ToString();
info["DataType"] = row.DataType.ToString();
}
else
{
var table = node as TableModel;
if (table != null)
{
info["Content"] = table.Content.ToString();
```

```
info["IsCollapsed"] = null;
info["MyIsCollapsed"] = table.IsCollapsed.ToString();
}
}

info["Position"] = string.Empty;
info["MyPosition"] = node.Position.ToInvariant();
}

public override NodeViewModelBase DeserializeNode(Telerik.Windows.Diagrams.Core.IShape
shape, Telerik.Windows.Diagrams.Core.SerializationInfo info)
{
NodeViewModelBase node = null;
if (shape is TableShape)
{
var table = new TableModel();
if (info["Content"] != null)
table.Content = info["Content"].ToString();
if (info["MyIsCollapsed"] != null)
table.IsCollapsed = bool.Parse(info["MyIsCollapsed"].ToString());
node = table;
}
else
{
var row = new RowModel();
if (info["ColumnName"] != null)
row.ColumnName = info["ColumnName"].ToString();
if (info["DataType"] != null)
row.DataType = (DataType)Enum.Parse(typeof(DataType), info["DataType"].ToString(), true);
node = row;
}
if (info["MyPosition"] != null)
node.Position = Utils.ToPoint(info["MyPosition"].ToString()).Value;

if (info[this.NodeUniqueIdKey] != null)
{
var nodeUniquekey = info[this.NodeUniqueIdKey].ToString();
if (!this.CachedNodes.ContainsKey(nodeUniquekey))
{
this.CachedNodes.Add(nodeUniquekey, node);
}
}
```

```
return node;
}

public override void AddNode(NodeViewModelBase node)
{
if (!this.InternalItems.Contains(node))
base.AddNode(node);
}
}
}
```

TableModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Telerik.Windows.Controls.Diagrams.Extensions.ViewModels;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class TableModel : ContainerNodeViewModelBase<NodeViewModelBase>
{
private bool isCollapsed;

public bool IsCollapsed
{
get
{
return this.isCollapsed;
}
set
{
if (this.isCollapsed != value)
{
this.isCollapsed = value;
this.OnPropertyChanged("IsCollapsed");
}
}
}
```

```
public override bool AddItem(object item)
{
var viewModel = item as RowModel;
if (viewModel != null)
{
viewModel.Position = new Point(this.Position.X, this.Position.Y + 90 + this.InternalItems.Count *
30);
return base.AddItem(item);
}

return false;
}

public override bool RemoveItem(object item)
{
if (item is NodeViewModelBase)
return base.RemoveItem(item);

return false;
}
}
}
```

TableShape.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles
{
public class TableShape : RadDiagramContainerShape
{
public int ContainerPosition { get; set; }

protected override bool IsDropPossible
{
get
```

```
{
return false;
}
}

protected override void OnItemsCollectionChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
{
base.OnItemsCollectionChanged(sender, e);
if (e.Action == System.Collections.Specialized.NotifyCollectionChangedAction.Remove)
{
var model = e.OldItems[0] as RowModel;
if (model == null) return;

foreach (var item in this.Items.OfType<RowModel>())
{
if (item.Position.Y >= model.Position.Y)
{
item.Position = new Point(item.Position.X, item.Position.Y - 30);
}
}
this.Height -= 30;
}
else if (e.Action == System.Collections.Specialized.NotifyCollectionChangedAction.Add)
{
var model = e.NewItems[0] as RowModel;
if (model == null) return;

if (this.Bounds.Bottom - 25 < model.Position.Y && (this.Diagram as
RadDiagram).UndoRedoService.IsActive)
this.ContentBounds = this.CalculateContentBounds(this.Bounds);

this.RefreshConnections();
}
}

private void RefreshConnections()
{
foreach (var connection in this.Diagram.Connections)
{
connection.Update();
}
```

```
}
}
}
```

CustomRadPane.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace Telerik.Windows.Controls
{
/// <summary>
/// Interaction logic for CustomRadPane.xaml
/// </summary>
public partial class CustomRadPane : UserControl
{
public CustomRadPane()
{
InitializeComponent();
}

public string FileName { get; set; }
}
}
```

DllFileInfo.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for DllFileInfo.xaml
/// </summary>
public partial class DllFileInfo : UserControl, IDLLInfoView
{
private DllFileInfoVM viewModelObj;

public DllFileInfo(DllFileInfoVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = dataContext;

((DllFileInfoVM)this.DataContext).View = this;
}
}
}
```

LicenceManager.xaml.cs

```
using System.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for LicenceManager.xaml
/// </summary>
```

```
public partial class LicenceManager : UserControl
{
public LicenceManager()
{
InitializeComponent();
}
}
}
```

PlaceHolderTextBox.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for PlaceHolderTextBox.xaml
/// </summary>
public partial class PlaceHolderTextBox : UserControl
{
public PlaceHolderTextBox()
{
InitializeComponent();
DataContext = new PlaceHolderTextBoxVM();
}
}
```

```
public class PlaceHolderTextBoxVM : ViewModelBase, IPVMLink
{

private PaneViewModel _pvm;
public PaneViewModel PVM
{
get { return _pvm; }
set
{
_pvm = value;
OnPropertyChanged("PVM");
this.PlaceHolderText = (value.DataObject as WaveformModel).FilePath;
}
}

private string _placeHolderText = string.Empty;
public string PlaceHolderText
{
get { return _placeHolderText; } set { _placeHolderText = value;
OnPropertyChanged("PlaceHolderText"); }
}

}
}
```

Splash.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Reflection;

namespace MT.TestStudio.GUI
{
```

```
/// <summary>
/// Interaction logic for Splash.xaml
/// </summary>
public partial class Splash : Window
{
#region Private member variables.

private static Splash splash = new Splash();

// To refresh the UI immediately
private delegate void RefreshDelegate();

#endregion Private member variables.

#region Constructor

/// <summary>
/// Constructor for splash class.
/// </summary>
public Splash()
{
InitializeComponent();

DataContext = this;
}

#endregion Constructor

#region Methods.

/// <summary>
/// Loads and displays the splash immage.
/// </summary>
public static void BeginDisplay()
{
splash.Show();
}

/// <summary>
/// Stops displaying the splash screen.
/// </summary>
public static void EndDisplay()
```

```
{
// Update GUI elements on the UI thread based on what instruments are in the system.
if (splash.statuslbl.Dispatcher.CheckAccess() == true)
{
Console.WriteLine("splash.statuslbl.Dispatcher.CheckAccess() == true");
SafeCloseSplash();
}
else
{
Console.WriteLine("splash.statuslbl.Dispatcher.CheckAccess() == false");
//splash.statuslbl.Dispatcher.Invoke(new SafeCloseGUIDelegate(SafeCloseSplash), new object[] {
});
}
}

public static void Loading(string test)
{
if (splash.statuslbl.Dispatcher.CheckAccess() == true)
{
SafeUpdateSplashLabel(test);
}
else
{
splash.statuslbl.Dispatcher.Invoke(new SafeUpdateGUIDelegate(SafeUpdateSplashLabel), new
object[] { test });
}
}

private delegate void SafeUpdateGUIDelegate(string message);

private static void SafeUpdateSplashLabel(string message)
{
splash.statuslbl.Content = message;
Refresh(splash.statuslbl);
}

private delegate void SafeCloseGUIDelegate();

private static void SafeCloseSplash()
{
splash.Close();
}
```

```
public static void HideDisplay()
{
// Update GUI elements on the UI thread based on what instruments are in the system.
if (splash.statuslbl.Dispatcher.CheckAccess() == true)
{
SafeHideSplash();
}
else
{
splash.statuslbl.Dispatcher.Invoke(new SafeCloseGUIDelegate(SafeHideSplash), new object[] { });
}
}

private static void SafeHideSplash()
{
splash.Hide();
}

public string Version
{
get
{
return "Version " + Assembly.GetExecutingAssembly().GetName().Version.ToString();
}
}

/// <summary>
/// Refresh
/// </summary>
/// <param name="obj"></param>
private static void Refresh(DependencyObject obj)
{
obj.Dispatcher.Invoke(System.Windows.Threading.DispatcherPriority.Render,
(RefreshDelegate)delegate { });
}

#endregion Methods.
}
}
```

UserPreferences.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models.AppModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for UserPreferences.xaml
/// </summary>
public partial class UserPreferences : UserControl
{
public UserPreferences(UserPreferencesData dataContext)
{
this.DataContext = dataContext;

InitializeComponent();
}

private void SavePreferencesButton_Click(object sender, RoutedEventArgs e)
{
try
{
(this.DataContext as
UserPreferencesData).SaveToXml(BackendConstants.UserPreferencesDataFilePath);
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
```

```
Console.WriteLine(ex.Message);
}
finally
{
RadWindow window = this.ParentOfType<RadWindow>();
window.Close();
}

}
}
}
```

WindowStyled.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Threading;

namespace MT.TestStudio.GUI
{
/// <summary>
/// Interaction logic for WindowStyled.xaml
/// </summary>
public partial class WindowStyled : Window
{
public WindowStyled(MainWindowViewModel MainWindowViewModel)
{
InitializeComponent();

// this.Resources.MergedDictionaries.Add(MainWindowViewModel.LanguageDict[0]);
```

```
// Subscribe to the VIEW's loaded event.
this.Loaded += new RoutedEventHandler(MainWindow_Loaded);
}

/// <summary>
/// Method that indicates to the view model that the view has been fully loaded.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
// To get the windows chrome ( top title bar ) to be styled we need
// to override the current style.
Style _style = null;

_style = (Style)Resources["CustomWindowStyle"];

this.Style = _style;
}

#region Public Methods

/// <summary>
/// Method that causes the window to close.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CloseClick(object sender, RoutedEventArgs e)
{
var window = (Window)((FrameworkElement)sender).TemplatedParent;
window.Close();
}

/// <summary>
/// Method that causes the window to be restored.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MaximizeRestoreClick(object sender, RoutedEventArgs e)
{
var window = (Window)((FrameworkElement)sender).TemplatedParent;
if (window.WindowState == System.Windows.WindowState.Normal)
```

```
{
window.WindowState = System.Windows.WindowState.Maximized;
}
else
{
window.WindowState = System.Windows.WindowState.Normal;
}
}

private void MinimizeClick(object sender, RoutedEventArgs e)
{
var window = (Window)((FrameworkElement)sender).TemplatedParent;
window.WindowState = System.Windows.WindowState.Minimized;
}

/// <summary>
/// Method that displays a dialog box.
/// </summary>
/// <param name="message"></param>
public void ShowDialog(string message)
{

// Check to see if we're on the UI thread before showing the dialog box.
if (Application.Current.Dispatcher.CheckAccess())
{
MessageBox.Show(Application.Current.MainWindow, message, "PXI Amplifier Test
Demonstration");
}
else
{
Application.Current.Dispatcher.Invoke(DispatcherPriority.Normal, new Action(() =>
{
MessageBox.Show(Application.Current.MainWindow, message, "PXI Amplifier Test
Demonstration");
}));
}
}

#endregion Public Methods
}
}
```

AttenuatorResultsViewer.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Microsoft.Win32;
using System;
using System.IO;
using System.Windows;
using System.Windows.Controls;
using Telerik.Charting;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for AttenuatorResultsViewer.xaml
/// </summary>
public partial class AttenuatorResultsViewer : UserControl
{
public AttenuatorResultsViewer(AttenuatorResultsViewerVM dataContext)
{
InitializeComponent();
//var result = (this.DataContext as PaneViewModel).PassedCalObject;
//this.DataContext = new AttenuatorResultsViewerVM(result as CalResult);
this.DataContext = dataContext;
}

/// <summary>
/// When you click on a data point in the grid view, the trackball moves to that point relative to the
chart.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RadGridView_SelectionChanged(object sender,
Telerik.Windows.Controls.SelectionChangeEventArgs e)
{
KvPOfDoubleDouble selected_kv = ChartDataGridView.SelectedItem as KvPOfDoubleDouble;
if ((bool)Y_Delta_Switch.IsChecked)
{
trackballBehavior.Position = Chart.ConvertDataToPoint(new DataTuple(selected_kv.Key,
selected_kv.Delta < 0 ? selected_kv.Delta * -1 : selected_kv.Delta));
}
else
{
```

```
trackballBehavior.Position = Chart.ConvertDataToPoint(new DataTuple(selected_kv.Key,
selected_kv.Value < 0 ? selected_kv.Value * -1 : selected_kv.Value));
}
}

/// <summary>
/// Export event for the gridview data.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RadButton_Click(object sender, RoutedEventArgs e)
{
var vm = this.DataContext as AttenuatorResultsViewerVM;
string extension = "csv";
SaveFileDialog dialog = new SaveFileDialog()
{
DefaultExt = extension,
Filter = String.Format("{1} files (.{0})|.{0}|All files (.)|.", extension, "Excel"),
FilterIndex = 1,
FileName = string.Format("{0}.{1}-{2}-{3}-{4}", vm.resultParentName, vm.resultName,
vm.SelectedFrequency, vm.SelectedAmpConfig, vm.SelectedGroup)
};
if (dialog.ShowDialog() == true)
{
using (Stream stream = dialog.OpenFile())
{
ChartDataGridView.Export(stream,
new GridViewExportOptions()
{
Format = ExportFormat.Csv,
ShowColumnHeaders = true,
ShowColumnFooters = true,
ShowGroupFooters = false,
});
}
}
}
}
}
```

CalDataInfo.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
```

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Windows;
using System.Windows.Controls;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for CalDataInfo.xaml
/// </summary>
public partial class CalDataInfo : UserControl
{
//Constructor for Previews window creation.
public CalDataInfo()
{
InitializeComponent();
}
//Constructor for Pane Creation.
public CalDataInfo(ResultDataVM dataContext)
{
InitializeComponent();
//var cdm = (this.DataContext as PaneViewModel).PassedCalObject;
this.DataContext = dataContext.CalDataInfo;
}

private void CalPointsGridView_AutoGeneratingColumn(object sender,
Telerik.Windows.Controls.GridViewAutoGeneratingColumnEventArgs e)
{
var dataColumn = e.Column as GridViewDataColumn;
if (dataColumn != null)
{
//Boolean columns
if (e.ItemPropertyInfo.PropertyType == typeof(bool) || e.ItemPropertyInfo.PropertyType ==
typeof(bool?))
{
GridViewCheckBoxColumn newColumn = new GridViewCheckBoxColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;
newColumn.Header = dataColumn.Header;
newColumn.UniqueName = dataColumn.UniqueName;
e.Column = newColumn;
}
```

```
//Double columns
if (e.ItemPropertyInfo.PropertyType == typeof(double))
{
if (e.Column.UniqueName == "Frequency" || e.Column.UniqueName == "Key")
{
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;
newColumn.Header = "Frequency";
newColumn.UniqueName = dataColumn.UniqueName;
e.Column = newColumn;
}
else
{
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;
newColumn.Header = dataColumn.Header;
newColumn.UniqueName = dataColumn.UniqueName;
newColumn.DataFormatString = "{0:N3}";
e.Column = newColumn;
}
}

//if (e.Column.UniqueName == "Value")
//{
//    e.Cancel = true;
//}
}
}

}
}
```

CalibrationConfiguratorView.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
```

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Unity;
using System.IO;
using Telerik.Windows.Data;
using MerlinTestStudio_Demo_Telerik.Data;
using System.Reflection;
using System.Windows.Input;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for CalibrationConfiguratorView.xaml
/// </summary>
public partial class CalibrationConfiguratorView : UserControl, MainView
{
private CalibrationConfiguratorVM viewModelObj;
private ObservableCollection<ContextMenuItem> rowContextMenuItems;
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

public CalibrationConfiguratorView(CalibrationConfiguratorVM dataContext)
{
InitializeComponent();
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<CalibrationConfiguratorVM>();
//OLD
this.DataContext = dataContext;

viewModelObj = dataContext;

viewModelObj.View = this as MainView;

//Enables row reorder behavior for the specified Gridview.
RowReorderBehavior.SetIsEnabled(CalPointGridView, true);
CalPointGridView.Drop += DropChanges;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.CalPointGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.CalPointGridView);
```

```
//Context menu items for CalPoint editor
ObservableCollection<ContextMenuItem> groupItems = new
ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Insert" },
new ContextMenuItem() { Text = "Delete" },
new ContextMenuItem() { Text = "Duplicate" }
};
this.rowContextMenuItems = groupItems;
}

private void DropChanges(object sender, DragEventArgs e)
{
viewModelObj.CurrentCalConfig.OnChangeOccured();
}

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
if (ClickedRow != null && ClickedRow.IsInEditMode == false) //ClickedRow.IsInEditMode == false;
Prevents major bug!
{
foreach (var item in this.rowContextMenuItems) { item.IsEnabled = true; }
GridContextMenu.ItemsSource = rowContextMenuItems;
if(!CalPointGridView.SelectedItems.Contains(ClickedRow.DataContext as CalPointModel))
{
CalPointGridView.SelectedCells.Clear();
CalPointGridView.SelectedItem = ClickedRow.DataContext as CalPointModel;
}
}
else
{
foreach (var item in this.rowContextMenuItems)
{
switch(item.Text)
{
case "Insert": item.IsEnabled = true; break;
case "Delete": item.IsEnabled = false; break;
case "Duplicate": item.IsEnabled = false; break;
}
}
GridContextMenu.ItemsSource = rowContextMenuItems;
}
```

```
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
switch (item.Text)
{
case "Insert": viewModelObj.InsertCalPointCommand.Execute(CalPointGridView.SelectedItems);
break;
case "Delete":
viewModelObj.RemoveCalPointsCommand.Execute(CalPointGridView.SelectedItems); break;
case "Duplicate":
viewModelObj.DuplicateCalPointCommand.Execute(CalPointGridView.SelectedItems); break;
}
}

/// <summary>
/// Utilizing the cell edit ended event to acheive multi-edit capability
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CalPointGridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
try
{
if (e.EditAction == GridViewEditAction.Commit) //&&  e.NewData != ""
{
string FirstPropertyChanged = e.Cell.Column.UniqueName;

List<Edit<int, object, string>> editedCalPoints = new List<Edit<int, object, string>>();
//Log the old value(s) and set the new value(s) of each CalPoint Selected.
for (int i = 0; i < CalPointGridView.SelectedCells.Count; i++)
{
GridViewCellInfo cell = CalPointGridView.SelectedCells[i];
object calPointModelObj = cell.Item;
string cellPropertyName = cell.Column.UniqueName;
object OldValue;

PropertyInfo firstProp = calPointModelObj.GetType().GetProperty(FirstPropertyChanged);
PropertyInfo cellProp = calPointModelObj.GetType().GetProperty(cellPropertyName);

if (firstProp.PropertyType == cellProp.PropertyType) //Protects against cross-selection type
```

```
conversion errors.
{
OldValue = cellProp.GetValue(calPointModelObj); //Get old value before it's overwrote.
if (i > 0) //Skip the first cell as the grid already commits the data.
{
cellProp.SetValue(calPointModelObj, e.NewData);
}

//This fixes a problem with the item ending the edit overwriting the old value with the new one.
if (calPointModelObj == e.Cell.ParentRow.Item) { OldValue = e.OldData; }

//Logs the index of the CalPoint edited and it's previous value.
editedCalPoints.Add(new Edit<int, object,
string>(CalPointGridView.Items.IndexOf(calPointModelObj), OldValue, cellPropertyName));
}
}

viewModelObj.CurrentCalConfig.LogEditAction(editedCalPoints);

//Forces OnChangesMade when the user changes the edits a CalPoint.
viewModelObj.CurrentCalConfig.OnChangeOccured();
}
}
catch(Exception ex) { MessageBox.Show(ex.Message); }
}

/// <summary>
/// Fires when the user hits the Delete key. This re-routes the action to the ViewModel so it can be
properly logged.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CalPointGridView_Deleting(object sender, GridViewDeletingEventArgs e)
{
viewModelObj.RemoveCalPointsCommand.Execute(CalPointGridView.SelectedItems);
e.Cancel = true;
}

#region MainView interface implementation
/// <summary>
/// Unselects CalPoint(s) when the user hits Esc key.
/// </summary>
```

```
public void UnselectCalPoints()
{
this.CalPointGridView.UnselectAll();
///try
///{
///    var scope = System.Windows.Input.FocusManager.GetFocusScope(CalPointGridView);
///    System.Windows.Input.FocusManager.SetFocusedElement(scope, null);
///    System.Windows.Input.Keyboard.ClearFocus();
///}
///catch (Exception ex) { MessageBox.Show(ex.Message); }
}

/// <summary>
/// Force the GridView to commit any pending edits.
/// </summary>
public void ForceCommitEdit()
{
this.CalPointGridView.CommitEdit();
}

/// <summary>
/// Used for undo functionality purposes. May not be needed.
/// </summary>
public void GridViewInvalidateMeasure()
{
this.CalPointGridView.ChildrenOfType<ScrollViewer>().First().InvalidateMeasure();
}
#endregion

//Fires per selected cell that the COPY action was performed on.
private void CalPointGridView_PastingCellClipboardContent(object sender,
GridViewCellClipboardEventArgs e)
{
try
{
List<Edit<int, object, string>> editedCalPoints = new List<Edit<int, object, string>>();
object calPointModelObj = e.Cell.Item;
string cellPropertyName = e.Cell.Column.UniqueName;

//Logs the index of the CalPoint edited and it's previous value.
object OldValue =
calPointModelObj.GetType().GetProperty(cellPropertyName).GetValue(calPointModelObj);
```

```
editedCalPoints.Add(new Edit<int, object,
string>(CalPointGridView.Items.IndexOf(calPointModelObj), OldValue, cellPropertyName));
viewModelObj.CurrentCalConfig.LogEditAction(editedCalPoints);

//Forces OnChangesMade when the user changes the edits a CalPoint.
viewModelObj.CurrentCalConfig.OnChangeOccured();
}
catch(Exception ex) { MessageBox.Show(ex.Message); };
}


private void CalPointGridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//This stops the user from selecting rows and accidently making a multi-edit on all the selected
cells within, also while preserving action logging for undo capability.
if (CalPointGridView.SelectedItems.Count() > 0)
{
var cell = CalPointGridView.CurrentCell;
CalPointGridView.SelectedItems.Clear();
CalPointGridView.SelectedCells.Add(new GridViewCellInfo(cell));
}
}

private void CalPointGridView_Loaded(object sender, RoutedEventArgs e)
{
//IsSynchronizedWithCurrentItem ="True", set CurrentItem to null so the first row isn't selected on
on load
CalPointGridView.CurrentItem = null;
}

private void UserControl_Unloaded(object sender, RoutedEventArgs e)
{
//viewModelObj = null; //Called when the user switched active panes.
}

}

#region ContextMenu MenuItemObjects

public class ContextMenuItem : INotifyPropertyChanged
{
```

```
private bool isEnabled = true;
private bool _isSeparator = false;
private string text;
private ObservableCollection<ContextMenuItem> subItems;

//Constructor should require a command to initialize an object. (if this is to become the standard for making context menus.
public ContextMenuItem()
{

}


public ICommand Command { get; set; }
//public object CommandParameter { get; set; }

public bool IsEnabled
{
get { return this.isEnabled; }
set
{
if (this.isEnabled != value)
{
this.isEnabled = value;
this.OnNotifyPropertyChanged("IsEnabled");
}
}
}
public bool IsSeparator
{
get { return this._isSeparator; }
set
{
if (this._isSeparator != value)
{
this._isSeparator = value;
this.OnNotifyPropertyChanged("IsSeparator");
}
}
}
public string Text
{
```

```
get { return this.text; }
set
{
if (this.text != value)
{
this.text = value;
this.OnNotifyPropertyChanged("Text");
}
}
}
public ObservableCollection<ContextMenuItem> SubItems
{
get
{
if (this.subItems == null)
{
this.subItems = new ObservableCollection<ContextMenuItem>();
}
return this.subItems;
}
set
{
if (this.subItems != value)
{
this.subItems = value;
this.OnNotifyPropertyChanged("SubItems");
}
}
}

#region INPC Members

public event PropertyChangedEventHandler PropertyChanged;
private void OnNotifyPropertyChanged(string ptopertyName)
{
if (this.PropertyChanged != null)
{
this.PropertyChanged(this, new PropertyChangedEventArgs(ptopertyName));
}
}

#endregion
```

```
}

#endregion
}
```

LimitUcEditor.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Converters;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.CalibrationViewModels;
using System;
using System.Collections.Generic;
using System.Dynamic;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Calibration
{
/// <summary>
/// Interaction logic for ProgramTestLimitEditor.xaml
/// </summary>
public partial class LimitUcEditor : UserControl, ILimitUcView
{
private LimitUcVM viewModelObj;

public LimitUcEditor(LimitUcVM dataContext)
{
InitializeComponent();

DataContext = dataContext;
```

```
viewModelObj = DataContext as LimitUcVM;

viewModelObj.View = this;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
//CAUSES BUG: Multi-Edit bug where the selected cell is the one being edited where as the move down cell is taking the new data... need fix to comply with current grid standards.
///this.LimitEditorGridView.KeyboardCommandProvider = new CustomKeyboardCommandProvider(this.LimitEditorGridView);


// this.LimitEditorGridView.RightFrozenColumnCount = 5;
this.LimitEditorGridView.RightFrozenColumnsSplitterVisibility = Visibility.Visible;
}


private void GridView_AutoGeneratingColumn(object sender, GridViewAutoGeneratingColumnEventArgs e)
{
var dataColumn = e.Column as GridViewDataColumn;

//Column nane specific, not type specific.
if (dataColumn != null)
{
string headerText;
Style CalStepCellStyle = Application.Current.FindResource("CalStepCellStyle") as Style;
Style PassFailCellStyle = Application.Current.FindResource("PassFailCellStyle") as Style;
Style AmplifiersCompressedCellStyle = Application.Current.FindResource("AmplifiersCompressedCellStyle") as Style;

switch (e.Column.UniqueName)
{
case "WasTargetLevelAcheived":
headerText = "Cal Step"; //headerText = "Pass/Fail";
dataColumn.DataMemberBinding.Converter = new BooleanToPassFailConverter();
e.Column = new GridViewDataColumn()
{
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None,
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
```

```
CellStyle = CalStepCellStyle
};
break;
case "AmplifiersCompressed": //Make sure name is correct.
headerText = "Amplifiers"; //headerText = "Amplifiers Compressed";
dataColumn.DataMemberBinding.Converter = new InverseBooleanToPassFailConverter();
e.Column = new GridViewDataColumn()
{
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None,
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
CellStyle = AmplifiersCompressedCellStyle
};
break;
case "DcAmplifierEnabled": //apart of Key object.
case "dcRfAmpEnabled": //apart of Value object.
headerText = dataColumn.Header.ToString();
dataColumn.DataMemberBinding.Converter = new BooleanToYesNoConverter();
e.Column = new GridViewDataColumn()
{
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None,
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
//CellStyle = PassFailCellStyle
};
break;
case "Frequency":
case "Key": //Same as below but with slightly different header (fast fix).
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None;
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;

#region Frequency Converter
//Set the converter for the column.
newColumn.DataMemberBinding.Converter = new FrequencyConverter();
newColumn.DataMemberBinding.ConverterParameter =
this.ViewingUnitComboBox.SelectedValue;
//newColumn.DataFormatString = "{0} " + this.ViewingUnitComboBox.SelectedValue; //Shows the
Unit right next to the data in each cell.
#endregion
```

```
newColumn.Header = "Frequency";
newColumn.UniqueName = dataColumn.UniqueName;
e.Column = newColumn;
break;
case "ifFreq":
GridViewDataColumn newIfFreqColumn = new GridViewDataColumn();
newIfFreqColumn.EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None;
newIfFreqColumn.DataMemberBinding = dataColumn.DataMemberBinding;

#region Frequency Converter
//Set the converter for the column.
newIfFreqColumn.DataMemberBinding.Converter = new FrequencyConverter();

//"{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:ResultDataView}}, Path=DataContext.ViewingUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
//var binding = new Binding("ViewingUnit")
//{
//    Mode = BindingMode.OneWay,
//    UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
//    RelativeSource = new RelativeSource(RelativeSourceMode.FindAncestor,
typeof(ResultDataView), 1),
//    Path = new PropertyPath("DataContext.ViewingUnit")
//};

//WORK AROUND. Needs new column generation as soon as SelectedValue is Changed, bad for
GUI, makes it slower.
newIfFreqColumn.DataMemberBinding.ConverterParameter =
this.ViewingUnitComboBox.SelectedValue;

newIfFreqColumn.DataFormatString = "{0} " + this.ViewingUnitComboBox.SelectedValue;
#endregion

newIfFreqColumn.Header = "If Frequency";
newIfFreqColumn.UniqueName = dataColumn.UniqueName;
e.Column = newIfFreqColumn;
break;
case "Min":
case "Max":
case "MinTolerance":
case "MaxTolerance":
```

```
case "Baseline":
//Create Columns w/ edit triggers.
GridViewDataColumn newColumn1 = new GridViewDataColumn() {  };
dataColumn.DataMemberBinding.Mode = BindingMode.TwoWay; //Bind Edit.
dataColumn.DataMemberBinding.UpdateSourceTrigger =
UpdateSourceTrigger.PropertyChanged; //Bind Edit.
newColumn1.DataMemberBinding = dataColumn.DataMemberBinding;
if(e.Column.UniqueName == "Baseline") { newColumn1.EditTriggers =
Telerik.Windows.Controls.GridView.GridViewEditTriggers.None; }
newColumn1.Header = dataColumn.Header;
newColumn1.UniqueName = dataColumn.UniqueName;
//if (e.Column.UniqueName == "Min" || e.Column.UniqueName == "Max") {
newColumn1.DataFormatString = "{0:N3}"; } //Visually fixes a calulation bug that produces too
many decimal places.
e.Column = newColumn1;
break;
default:
GridViewDataColumn newColumn2 = new GridViewDataColumn();
newColumn2.DataMemberBinding = dataColumn.DataMemberBinding;
newColumn2.EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None;
///switch (dataColumn.UniqueName)
///{
///    case "GainCPLBoost": newColumn.Header = "RF110 A1"; break;
///    case "GainLnaA": newColumn.Header = "RF110 A2"; break;
///    case "GainLnaB": newColumn.Header = "RF110 A3"; break;
///    case "GainA1": newColumn.Header = "RF210 A1"; break;
///    case "GainA2": newColumn.Header = "RF210 A2"; break;
///    case "GainA3": newColumn.Header = "RF210 A3"; break;
///}
newColumn2.Header = dataColumn.Header;
newColumn2.UniqueName = dataColumn.UniqueName;
//newColumn2.DataFormatString = "{0:N3}";
e.Column = newColumn2;
///Previous default Bool Column code.
//headerText = dataColumn.Header.ToString();
//e.Column = new GridViewCheckBoxColumn()
//{
//    EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None,
//    DataMemberBinding = dataColumn.DataMemberBinding,
//    Header = headerText,
//    UniqueName = dataColumn.UniqueName,
//    CellStyle = PassFailCellStyle
```

```
//};
break;
}

if (e.Column.UniqueName == "Value")
{
e.Cancel = true;
}
}
}

//WORK AROUND
//On viewing unit selection changed, update the autogenerated columns by forcing column generation to update
private void ViewingUnitComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
this.LimitEditorGridView.AutoGenerateColumns = false;
this.LimitEditorGridView.AutoGenerateColumns = true;
}

public void ForceCommitEdit()
{
this.LimitEditorGridView.CommitEdit();
}

//May not be needed.
public void RebindGridView()
{
this.LimitEditorGridView.Rebind();
}

private void LimitEditorGridView_CellValidating(object sender, GridViewCellValidatingEventArgs e)
{
try
{
double newDataParsed = double.Parse(e.NewValue.ToString());
dynamic DynExObj = e.Row.Item;
double Baseline = (double)DynExObj.Baseline;

switch (e.Cell.Column.UniqueName)
```

```
{
case "Min":
if (newDataParsed >= Baseline)
{
e.IsValid = false;
e.ErrorMessage = "Min cannot be greater than or equal to the Baseline.";
}
else { e.IsValid = true; }
break;
case "Max":
if (newDataParsed <= Baseline)
{
e.IsValid = false;
e.ErrorMessage = "Max cannot be less than or equal to the Baseline.";
}
else { e.IsValid = true; }
break;
case "MinTolerance":
case "MaxTolerance":
if (newDataParsed < 0) //Check for negative value.
{
e.IsValid = false;
e.ErrorMessage = "Tolerance cannot be set as a negative value.";
}
else { e.IsValid = true; }
break;
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }

}

private void LimitEditorGridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs
e)
{
try
{
if (e.EditAction == GridViewEditAction.Commit) //&&  e.NewData != ""
{
//NaN Check, don't allow NaN as an input.
if(e.NewData is double && double.IsNaN((double)e.NewData) ) //TEST
{
```

```
Console.WriteLine("User input 'NaN' is not allowed.");
//e.Handled = true;
return; //Stop the code, do not commit edit to object
}

for (int i = 0; i < this.LimitEditorGridView.SelectedCells.Count; i++)
{
GridViewCellInfo cell = this.LimitEditorGridView.SelectedCells[i];

dynamic DynExObj = cell.Item;
CalculateDynamicExpandoObjectProperties(ref DynExObj, cell.Column.UniqueName,
e.NewData);
}

viewModelObj.CalLimits.OnChangeOccured();
}
}
catch (Exception ex) { MessageBox.Show(ex.Message);  Console.WriteLine(ex.Message); }
}

private void LimitEditorGridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//This stops the user from selecting rows and accidently making a multi-edit on all the selected
cells within (also while preserving action logging for undo capability.)
if (this.LimitEditorGridView.SelectedItems.Count() > 0)
{
var cell = this.LimitEditorGridView.CurrentCell;
this.LimitEditorGridView.SelectedItems.Clear();
this.LimitEditorGridView.SelectedCells.Add(new GridViewCellInfo(cell));
}
}

//Fires per selected cell that the paste action is performed on.
private void LimitEditorGridView_PastingCellClipboardContent(object sender,
GridViewCellClipboardEventArgs e)
{
try
{
dynamic DynExObj = e.Cell.Item;
CalculateDynamicExpandoObjectProperties(ref DynExObj, e.Cell.Column.UniqueName, e.Value);
```

```
viewModelObj.CalLimits.OnChangeOccured();
}
catch (Exception ex) { MessageBox.Show(ex.Message); };
}

private void CalculateDynamicExpandoObjectProperties(ref dynamic DynExObj, string
propertyName, object NewData)
{
double newDataParsed = double.Parse(NewData.ToString());
double Baseline = (double)DynExObj.Baseline;

switch (propertyName)
{
case "Min":
double minTolNan = double.Parse(DynExObj.MinTolerance.ToString());
if (!Double.IsNaN(minTolNan))
{
DynExObj.Min = newDataParsed;
DynExObj.MinTolerance = Math.Abs(Baseline - DynExObj.Min);
}
else { DynExObj.Min = double.NaN; }
break;
case "Max":
double maxTolNaN = double.Parse(DynExObj.MaxTolerance.ToString());
if (!Double.IsNaN(maxTolNaN))
{
DynExObj.Max = newDataParsed;
DynExObj.MaxTolerance = Math.Abs(DynExObj.Max - Baseline);
}
else { DynExObj.Max = double.NaN; }
break;
case "MinTolerance":
double minNaN = double.Parse(DynExObj.Min.ToString());
if (!Double.IsNaN(minNaN))
{
DynExObj.MinTolerance = Math.Abs(newDataParsed);
DynExObj.Min = Baseline - DynExObj.MinTolerance;
}
else { DynExObj.MinTolerance = double.NaN; }
break;
case "MaxTolerance":
double maxNaN = double.Parse(DynExObj.Max.ToString());
```

```
if (!Double.IsNaN(maxNaN))
{
DynExObj.MaxTolerance = Math.Abs(newDataParsed);
DynExObj.Max = Baseline + DynExObj.MaxTolerance;
}
else { DynExObj.MaxTolerance = double.NaN; }
break;
}

}

}
}
```

ResultDataView.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Converters;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for ResultDataView.xaml
/// </summary>
```

```
public partial class ResultDataView : UserControl
{
public ResultDataView(ResultDataVM dataContext)
{
InitializeComponent();
//var result = (this.DataContext as PaneViewModel).PassedCalObject;
//this.DataContext = new ResultDataVM(result as CalResult);
this.DataContext = dataContext;
}

/// <summary>
/// Export event for the gridview data.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RadMenuItem_Click(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
var vm = this.DataContext as ResultDataVM;
string extension = "csv";
SaveFileDialog dialog = new SaveFileDialog()
{
DefaultExt = extension,
Filter = String.Format("{1} files (.{0})|.{0}|All files (.)|.", extension, "Excel"),
FilterIndex = 1,
FileName = string.Format("{0}.{1}", vm.resultParentName, vm.resultName)
};
if (dialog.ShowDialog() == true)
{
//before your loop
var csv = new StringBuilder();

foreach(var resultItem in vm.GridViewData)
{
string lineToAppend = string.Empty;
foreach(PropertyInfo prop in resultItem.GetType().GetProperties())
{
lineToAppend += prop.GetValue(resultItem);
lineToAppend += ",";
}
csv.AppendLine(lineToAppend);
```

```
}

//after your loop
File.WriteAllText(dialog.FileName, csv.ToString());

}
}
catch(Exception ex ) { MessageBox.Show(ex.Message); }
}

private void CalResultGridView_AutoGeneratingColumn(object sender,
GridViewAutoGeneratingColumnEventArgs e)
{
var dataColumn = e.Column as GridViewDataColumn;
if (dataColumn != null)
{
//Boolean columns
if (e.ItemPropertyInfo.PropertyType == typeof(bool) || e.ItemPropertyInfo.PropertyType ==
typeof(bool?))
{
string headerText;
Style CalStepCellStyle = Application.Current.FindResource("CalStepCellStyle") as Style;
Style PassFailCellStyle = Application.Current.FindResource("PassFailCellStyle") as Style;
Style AmplifiersCompressedCellStyle =
Application.Current.FindResource("AmplifiersCompressedCellStyle") as Style;

switch (e.Column.UniqueName)
{
case "WasTargetLevelAcheived":
headerText = "Cal Step"; //headerText = "Pass/Fail";
dataColumn.DataMemberBinding.Converter = new BooleanToPassFailConverter();
e.Column = new GridViewDataColumn()
{
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
CellStyle = CalStepCellStyle
};
break;
case "AmplifiersCompressed": //Make sure name is correct.
headerText = "Amplifiers"; //headerText = "Amplifiers Compressed";
dataColumn.DataMemberBinding.Converter = new InverseBooleanToPassFailConverter();
```

```
e.Column = new GridViewDataColumn()
{
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
CellStyle = AmplifiersCompressedCellStyle
};
break;
case "PassLimitStatus":
case "LimitPassStatus":
//Change on HOLD.
//Some how transfer value to CalFactor (if applicable) for red hightlight (if failed)
e.Cancel = true; //Don't generate this column.
break;
case "ResultPass":
headerText = dataColumn.Header.ToString();
dataColumn.DataMemberBinding.Converter = new BooleanToYesNoConverter();
e.Column = new GridViewDataColumn()
{
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
//CellStyle = PassFailCellStyle
};
break;
case "DcAmplifierEnabled": //apart of Key object.
case "dcRfAmpEnabled": //apart of Value object.
headerText = dataColumn.Header.ToString();
dataColumn.DataMemberBinding.Converter = new BooleanToYesNoConverter();
e.Column = new GridViewDataColumn()
{
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
//CellStyle = PassFailCellStyle
};
break;
default: //DEFAULT
headerText = dataColumn.Header.ToString();
e.Column = new GridViewCheckBoxColumn()
{
DataMemberBinding = dataColumn.DataMemberBinding,
```

```
Header = headerText,
UniqueName = dataColumn.UniqueName,
CellStyle = PassFailCellStyle
};
break;
}

}

#region Enum Column Comments
///Enum type fields get generated as GridViewDataColumn(s).
//if (e.ItemPropertyInfo.PropertyType == typeof(Enum))
//{
//    GridViewDataColumn newColumn = new GridViewDataColumn();
//    newColumn.DataMemberBinding = dataColumn.DataMemberBinding;
//    newColumn.Header = "Frequency";
//    newColumn.UniqueName = dataColumn.UniqueName;
//    e.Column = newColumn;
//}
#endregion

//Double columns
if (e.ItemPropertyInfo.PropertyType == typeof(double))
{
if (e.Column.UniqueName == "Frequency" || e.Column.UniqueName == "Key")
{
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;

#region Frequency Converter
//Set the converter for the column.
newColumn.DataMemberBinding.Converter = new FrequencyConverter();

//"{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:ResultDataView}}, Path=DataContext.ViewingUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
//var binding = new Binding("ViewingUnit")
//{
//    Mode = BindingMode.OneWay,
//    UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
//    RelativeSource = new RelativeSource(RelativeSourceMode.FindAncestor,
typeof(ResultDataView), 1),
```

```
//    Path = new PropertyPath("DataContext.ViewingUnit")
//};

//WORK AROUND. Needs new column generation as soon as SelectedValue is Changed, bad for GUI, makes it slower.
newColumn.DataMemberBinding.ConverterParameter =
this.ViewingUnitComboBox.SelectedValue;

//newColumn.DataFormatString = "{0} " + this.ViewingUnitComboBox.SelectedValue; //Shows the Unit right next to the data in each cell.
#endregion

newColumn.Header = "Frequency";
newColumn.UniqueName = dataColumn.UniqueName;
e.Column = newColumn;
}
else if(e.Column.UniqueName == "ifFreq")
{
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;

//Same code as above...
#region Frequency Converter
//Set the converter for the column.
newColumn.DataMemberBinding.Converter = new FrequencyConverter();
newColumn.DataMemberBinding.ConverterParameter =
this.ViewingUnitComboBox.SelectedValue;
newColumn.DataFormatString = "{0} " + this.ViewingUnitComboBox.SelectedValue;
#endregion

newColumn.Header = "If Frequency";
newColumn.UniqueName = dataColumn.UniqueName;
e.Column = newColumn;
}
else
{
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;
switch (dataColumn.UniqueName)
{
case "GainCPLBoost": newColumn.Header = "RF110 A1"; break;
case "GainLnaA": newColumn.Header = "RF110 A2"; break;
```

```
case "GainLnaB": newColumn.Header = "RF110 A3"; break;
case "GainA1": newColumn.Header = "RF210 A1"; break;
case "GainA2": newColumn.Header = "RF210 A2"; break;
case "GainA3": newColumn.Header = "RF210 A3"; break;
}
//newColumn.Header = dataColumn.Header;
newColumn.UniqueName = dataColumn.UniqueName;
newColumn.DataFormatString = "{0:N3}";
e.Column = newColumn;
}
}

if (e.Column.UniqueName == "Value")
{
e.Cancel = true; //Don't generate this column.
}
}
}

//WORK AROUND
//On viewing unit selection changed, update the autogenerated columns by forcing column generation to update
private void ViewingUnitComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
this.CalResultGridView.AutoGenerateColumns = false;
this.CalResultGridView.AutoGenerateColumns = true;
}
}
}
```

ProductConfigurationView.xaml.cs

```
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
```

```
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for ProductConfiguration.xaml
/// </summary>
public partial class ProductConfigurationView : UserControl, IProductConfigView
{
private ProductConfigViewModel viewModelObj;

public ProductConfigurationView(ProductConfigViewModel dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = dataContext;

((ProductConfigViewModel)this.DataContext).View = this;
}



}
}
```

ProjectConfiguration.xaml.cs

```
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
```

```
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Unity;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for ProjectConfiguration.xaml
/// </summary>
public partial class ProjectConfiguration : UserControl
{
public ProjectConfiguration()
{
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<ProjectConfigViewModel>();

this.DataContext = new ProjectConfigViewModel();

InitializeComponent();

}

private void RadComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
if(this.IsLoaded)
{
Console.WriteLine("Project target system has been changed, press save all to save changes...");
}
//if (e.AddedItems.Count > 0)
//{
//    Console.WriteLine("Project target system has been changed, press save all to save
changes...");
//}
}
}
}
```

SystemConfiguration.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Windows;
```

```
using System.Windows.Controls;
using Unity;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for SystemConfiguration.xaml
/// </summary>
public partial class SystemConfiguration : UserControl
{
public SystemConfiguration()
{
DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<SystemConfigViewModel>();

InitializeComponent();
}
}
}
```

ConnectionDiagram.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for ConnectionDiagram.xaml
/// </summary>
public partial class ConnectionDiagram : UserControl
```

```
{
public ConnectionDiagram()
{
InitializeComponent();
}
}
}
```

Digital_Cable_Map.xaml.cs

```
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows;
using System.Windows.Input;
using Telerik.Windows.Diagrams.Core;
using MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Diagrams.Extensions.ViewModels;
using System;
using System.Windows.Threading;

namespace MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager
{
/// <summary>
/// Interaction logic for Digital_Cable_Map.xaml
/// </summary>
public partial class Digital_Cable_Map : UserControl
{
private Telerik.Windows.Controls.Diagrams.Extensions.FileManager fileManager;
private TablesGraphSource dc;
private bool isClear;

static Digital_Cable_Map()
{
var saveBinding = new CommandBinding(DiagramCommands.Save, ExecuteSave,
CanExecutedSave);
var openBinding = new CommandBinding(DiagramCommands.Open, ExecuteOpen);

CommandManager.RegisterClassCommandBinding(typeof(Digital_Cable_Map), saveBinding);
CommandManager.RegisterClassCommandBinding(typeof(Digital_Cable_Map), openBinding);
}
```

```
public Digital_Cable_Map()
{
DiagramConstants.RoutingGridSize = 40d;
DiagramConstants.ContainerMargin = 0d;
InitializeComponent();

this.fileManager = new
Telerik.Windows.Controls.Diagrams.Extensions.FileManager(this.diagram);
diagram.AddShape(new TableShape() { Width = 100, Height = 50, Position = new Point(0, 0) });
diagram.AddShape(new TableShape() { Width =100, Height = 50, Position = new Point(0, 0) });
this.DataContext = this.dc = new TablesGraphSource();
var newRouter = new AStarRouter(this.diagram) { WallOptimization = true };
this.diagram.RoutingService.Router = newRouter;
this.diagram.RoutingService.FreeRouter = newRouter;
this.Loaded += this.OnLoaded;
}

private static void CanExecutedSave(object sender, CanExecuteRoutedEventArgs e)
{
var owner = sender as Digital_Cable_Map;
e.CanExecute = owner != null && owner.diagram != null && owner.diagram.Items.Count > 0;
}

private static void ExecuteSave(object sender, ExecutedRoutedEventArgs e)
{
var owner = sender as Digital_Cable_Map;
if (owner != null)
owner.fileManager.SaveToFile();
}

private static void ExecuteOpen(object sender, ExecutedRoutedEventArgs e)
{
var owner = sender as Digital_Cable_Map;
if (owner != null)
{
owner.ClearDiagram();
owner.fileManager.LoadFromFile();
}
}

private void OnConnectionManipulationCompleted(object sender, ManipulationRoutedEventArgs
```

```
e)
{
if (e.ManipulationPoint.Type != ManipulationPointType.Intermediate)
e.Handled = e.Shape == null;
}

private void OnSelectionChanged(object sender, SelectionChangedEventArgs e)
{
if (e.AddedItems.Count > 0)
{
foreach (var container in this.diagram.Shapes.OfType<RadDiagramContainerShape>())
{
if (container.IsSelected)
container.ZIndex = 4;
else
container.ZIndex = 0;
}
}
}

private void OnPreviewSelectionChanged(object sender, SelectionChangedEventArgs e)
{
if (e.AddedItems.Count > 1)
{
foreach (var item in e.AddedItems)
{
if (item is RowModel) continue;

var container = this.diagram.ContainerGenerator.ContainerFromItem(item);
container.IsSelected = true;
}
e.Handled = true;
}
}

private void OnNewClick(object sender, RoutedEventArgs e)
{
this.RefreshDiagram();
}

private void OnLoaded(object sender, RoutedEventArgs routedEventArgs)
{
```

```
this.RefreshDiagram();
}

private void LoadDefaultTables()
{
//SamplesFactory.LoadSample(this.diagram, "tableShape");
}

private void ClearDiagram()
{
this.diagram.UndoRedoService.Clear();
var graphSource = this.DataContext as TablesGraphSource;
if (graphSource != null)
{
graphSource.ClearCache();
graphSource.Clear();
}
}

private void OnItemsChanging(object sender, DiagramItemsChangingEventArgs e)
{
if (this.isClear) return;

if (e.Action == System.Collections.Specialized.NotifyCollectionChangedAction.Remove)
{
var rowModel = e.OldItems.ElementAt(0) as RowModel;
var command = new CompositeCommand("Remove Connections");
if (rowModel != null)
{
this.RemoveRowModel(rowModel, command);
if (command.Commands.Count() > 0)
this.diagram.UndoRedoService.ExecuteCommand(command);
}
else
{
var tableModel = e.OldItems.ElementAt(0) as TableModel;
if (tableModel != null)
{
foreach (var item in tableModel.InternalItems)
{
this.RemoveRowModel(item as RowModel, command);
}
```

```
if (command.Commands.Count() > 0)
this.diagram.UndoRedoService.ExecuteCommand(command);

tableModel.InternalItems.Clear();
}
}
}
}

private void RemoveRowModel(RowModel rowModel, CompositeCommand command)
{
var container = this.diagram.ContainerGenerator.ContainerFromItem(rowModel) as IShape;
if (container == null) return;

foreach (var connection in this.diagram.GetConnectionsForShape(container).ToList())
{
var link = this.diagram.ContainerGenerator.ItemFromContainer(connection) as
LinkViewModelBase<NodeViewModelBase>;
command.AddCommand(new UndoableDelegateCommand(
"Remove link",
new Action<object>((o) => this.dc.RemoveLink(link)),
new Action<object>((o) => this.dc.AddLink(link))));
}
}

private void RefreshDiagram()
{
this.isClear = true;
this.diagram.DeselectAll();
this.ClearDiagram();
this.LoadDefaultTables();
this.isClear = false;
}

private void OnDiagramDeserialized(object sender, RadRoutedEventArgs e)
{
// We need to update the connections after the containers have been collapsed.
this.Dispatcher.BeginInvoke(new Action(() =>
{
foreach (var connection in this.diagram.Connections)
{
connection.Update();
```

```
}

}), DispatcherPriority.ApplicationIdle);
}
}
}
```

PinMap.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Reflection;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
using Unity;

namespace MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager
{
/// <summary>
/// Interaction logic for PinMap.xaml
/// </summary>
public partial class PinMap : UserControl, IPinMapView
{
private PinMapViewModel viewModelObj;
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private ObservableCollection<ContextMenuItem> ContextMenuItems = new
ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Insert" },
new ContextMenuItem() { Text = "Delete" }
//new MenuItem() { Text = "Duplicate" }
};
```

```
public PinMap(object dataContext)
{
InitializeComponent();

///Set the source then set selected maybe...
PinMapSectionListBox.ItemsSource = new List<string>
{
"RF Pins",
"Digital Pins",
"Power Supply Pins"
};
//PinMapSectionListBox.SelectedItem = PinMapSectionListBox.Items[0];

//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<PinMapViewModel>();
this.DataContext = dataContext;
viewModelObj = dataContext as PinMapViewModel;
viewModelObj.View = this as IPinMapView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.PinMapGrid.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.PinMapGrid);

PinMapGrid.Deleting += PinMapGrid_Deleting;
PinMapGrid.CellEditEnded += PinMapGrid_CellEditEnded;

//Needs IView and viewModelObj property upon reconstruction.
}

#region Context Menu
private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
if (ClickedRow != null && ClickedRow.IsInEditMode == false) //ClickedRow.IsInEditMode == false;
Prevents major bug!
{
foreach (var item in this.ContextMenuItems) { item.IsEnabled = true; }
GridContextMenu.ItemsSource = ContextMenuItems;
if (!PinMapGrid.SelectedItems.Contains(ClickedRow.DataContext as Data.PinModel))
{
PinMapGrid.SelectedCells.Clear();
PinMapGrid.SelectedItem = ClickedRow.DataContext as Data.PinModel;
}
```

```
}
else
{
//foreach (var item in this.ContextMenuItems)
//{
//    switch (item.Text)
//    {
//        case "Insert": item.IsEnabled = true; break;
//        case "Delete": item.IsEnabled = false; break;
//        //case "Duplicate": item.IsEnabled = false; break;
//    }
//}
//GridContextMenu.ItemsSource = ContextMenuItems;
GridContextMenu.ItemsSource = null;
}
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
switch (item.Text)
{
case "Insert": viewModelObj.AddPinCommand.Execute(PinMapGrid.SelectedItems); break;
case "Delete": viewModelObj.RemovePinCommand.Execute(PinMapGrid.SelectedItems); break;
//case "Duplicate": viewModelObj.DuplicatePinCommand.Execute(PinMapGrid.SelectedItems);
break;
}
}
#endregion
private void PinMapGrid_Deleting(object sender, GridViewDeletingEventArgs e)
{
viewModelObj.RemovePinCommand.Execute((sender as RadGridView).SelectedItems);
e.Cancel = true;
}
private void PinMapGrid_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
#region PinMap Multi Edit
try
{
if (e.EditAction == GridViewEditAction.Commit)
{
string FirstPropertyChanged = e.Cell.Column.UniqueName.Split(' ')[0];
```

```
//Log the old value(s) and set the new value(s) of each CalPoint Selected.
for (int i = 0; i < PinMapGrid.SelectedCells.Count; i++)
{
GridViewCellInfo cell = PinMapGrid.SelectedCells[i];
GridViewColumnGroup group = PinMapGrid.ColumnGroups.First(g => g.Name ==
cell.Column.ColumnGroupName);
int siteIndex = PinMapGrid.ColumnGroups.IndexOf(group) - 1; // - 1 due to the PIN INFO column
group.

if (siteIndex >= 0) //Is Site Column Group
{
Data.PinModel pin = cell.Item as Data.PinModel;
Data.MappingModel mapping = pin.mappings[siteIndex];

string cellPropertyName = cell.Column.UniqueName.Split(' ')[0];
object OldValue;

//PropertyInfo firstProp = mapping.GetType().GetProperty(FirstPropertyChanged); //Different
objects pin and mapping null comparison for properties.
PropertyInfo cellProp = mapping.GetType().GetProperty(cellPropertyName);

if (FirstPropertyChanged == cellPropertyName)
{
OldValue = cellProp.GetValue(mapping); //Get old value before it's overwrote.
if (i > 0) //Skip the first cell as the grid already commits the data.
{
cellProp.SetValue(mapping, e.NewData);
}

//This fixes a problem with the item ending the edit overwriting the old value with the new one.
if (pin == e.Cell.ParentRow.Item) { OldValue = e.OldData; }
}
}
else //Is Pin Info Column group, Use as Pin Obj
{
object pinObj = cell.Item;
string cellPropertyName = cell.Column.UniqueName;
object OldValue;

//PropertyInfo firstProp = pinObj.GetType().GetProperty(FirstPropertyChanged); //Different objects
pin and mapping null comparison for properties.
```

```
PropertyInfo cellProp = pinObj.GetType().GetProperty(cellPropertyName);

if (FirstPropertyChanged == cellPropertyName)
{
OldValue = cellProp.GetValue(pinObj); //Get old value before it's overwrote.
if (i > 0) //Skip the first cell as the grid already commits the data.
{
cellProp.SetValue(pinObj, e.NewData);
}

//This fixes a problem with the item ending the edit overwriting the old value with the new one.
if (pinObj == e.Cell.ParentRow.Item) { OldValue = e.OldData; }
}
}
}

//Forces OnChangesMade when the user changes the edits a CalPoint.
viewModelObj.OnChangeOccured();
}
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine("Error occured executing a pin map multi-edit action.");
}

#endregion
}

public RadGridView GetGridView()
{
return this.PinMapGrid;
}

public void UnselectAll()
{
this.PinMapGrid.UnselectAll();
}

public void ForceCommitEdit()
{
this.PinMapGrid.CommitEdit();
```

```
}

private void PinMapGrid_PreparedCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//This stops the user from selecting rows and accidently making a multi-edit on all the selected
cells within, also while preserving action logging for undo capability.
if (PinMapGrid.SelectedItems.Count() > 0)
{
var cell = PinMapGrid.CurrentCell;
PinMapGrid.SelectedItems.Clear();
PinMapGrid.SelectedCells.Add(new GridViewCellInfo(cell));
}
}


//IN PROGRESS
private void PinMapGrid_CellValidating(object sender, GridViewCellValidatingEventArgs e)
{
try
{
//double newDataParsed = double.Parse(e.NewValue.ToString());
//dynamic DynExObj = e.Row.Item;
//double Baseline = (double)DynExObj.Baseline;
//
//switch (e.Cell.Column.UniqueName)
//{
//    case "Min":
//        if (newDataParsed >= Baseline)
//        {
//            e.IsValid = false;
//            e.ErrorMessage = "Min cannot be greater than or equal to the Baseline.";
//        }
//        else { e.IsValid = true; }
//        break;
//    case "Max":
//        if (newDataParsed <= Baseline)
//        {
//            e.IsValid = false;
//            e.ErrorMessage = "Max cannot be less than or equal to the Baseline.";
//        }
//        else { e.IsValid = true; }
```

```
//        break;
//    case "MinTolerance":
//    case "MaxTolerance":
//        if (newDataParsed < 0) //Check for negative value.
//        {
//            e.IsValid = false;
//            e.ErrorMessage = "Tolerance cannot be set as a negative value.";
//        }
//        else { e.IsValid = true; }
//        break;
//}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
}
}
```

RF_Cable_Map.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.GraphViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows.Controls.Diagrams.Extensions.ViewModels;
using Telerik.Windows.Diagrams.Core;

namespace MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager
{
/// <summary>
/// Interaction logic for RF_Cable_Map.xaml
/// </summary>
public partial class RF_Cable_Map : UserControl, IDiagramView
{
private DiagramViewModel viewModelObj;
//DiagramViewModel DVM = new DiagramViewModel();
//List<RadDiagramConnector> connectorList = new List<RadDiagramConnector>();

public RF_Cable_Map(DiagramViewModel dataContext)
```

```
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = dataContext;

((DiagramViewModel)this.DataContext).View = this;

if(diagram.Items.Count == 0)
{
this.diagram.GraphSource = new GraphSource();
}

//tree.ItemsSource = diagram.GraphSource.Items;

this.diagram.Loaded += ContainerGenerator_StatusChanged;
}

//Saves the diagram to a string so we can write it to an XML file.
public string GetDiagramSerString()
{
return this.diagram.Save();
}

//Loads the diagram by setting a Deserialization string.
public void SetDiagramDeSerString(string DeSerString)
{
this.diagram.Load(DeSerString);
}

//public GraphSource GetGraphSource()
//{
//    return this.diagram.GraphSource as GraphSource;
//}
//
//public void SetGraphSource(GraphSource graphSourceToSet)
//{
//    if (graphSourceToSet != null)
//    {
//        diagram.GraphSource = graphSourceToSet;
//    }
```

```
//    else { Console.WriteLine("Cannot set GraphSource to null"); }
//
//}

private void ContainerGenerator_StatusChanged(object sender, EventArgs e)
{
if (this.diagram.ContainerGenerator.Status == GeneratorStatus.ContainersGenerated)
{
foreach (object node in diagram.GraphSource.Items)
{
if (node is Port)
{
var shape = diagram.ContainerGenerator.ContainerFromItem(node) as RadDiagramShape;
if (shape != null)
{
shape.Connectors.Clear();
var connectorCollection = diagram.Resources["SecondCollection"] as
CustomConnectorCollection;
if (connectorCollection != null)
{
foreach (RadDiagramConnector connector in connectorCollection)
{
var newConnector = new RadDiagramConnector() { Offset = connector.Offset, Name =
connector.Name + "_" + shape.GetHashCode().ToString() };
shape.Connectors.Add(newConnector);
}
}
}
}
}
}
}

private void diagram_ConnectorActivationChanged(object sender,
ConnectorActivationChangedEventArgs e)
{
//viewModelObj.OnChangeOccured();
}

private void diagram_ItemsChanged(object sender, DiagramItemsChangedEventArgs e)
{
//viewModelObj.OnChangeOccured();
```

```
}

private void diagram_PositionChanged(object sender, PositionChangedRoutedEventArgs e)
{
//viewModelObj.OnChangeOccured();
}

//private void InitializeConnections()
//{
//    int Name = 0; // temp int for naming
//    Diagram.Shapes.ToList().ForEach(x =>
//    {
//        //x.Connectors.Clear();
//
//        int n = 0;
//        double d = 0;
//        var i = 0.1;
//        while (n != 10 )
//        {
//            RadDiagramConnector a = new RadDiagramConnector() { Offset = new Point(1, d),
Name = ("Shape" + Name.ToString() + "Connector" + n.ToString())};
//
//            connectorList.Add(a);
//            d = (d + i);
//            ++n;
//        }
//
//        foreach (RadDiagramConnector RDC in connectorList)
//        {
//            x.Connectors.Add(RDC);
//        }
//
//        ++Name;
//    });
//}
}
}
```

AboutBox.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Diagnostics;
using System.Windows.Navigation;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for AboutBox.xaml
/// </summary>
public partial class AboutBox : Window
{
/// <summary>
/// Constructor for the About Box Window.
/// </summary>
public AboutBox()
{
InitializeComponent();

// Set the XAML's data context to be this class.
DataContext = this;
}

/// <summary>
/// Gets the current version of the GUI assembly.
/// </summary>
public string Version
{
get
{
return Assembly.GetExecutingAssembly().GetName().Version.ToString();
}
}
```

```
/// <summary>
/// Method that's invoked when the 'OK ' button is clicked and closes the about box.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void okButton_Click(object sender, RoutedEventArgs e)
{
this.Close();
}

/// <summary>
/// Method that's invoked when the link in the about box is clicked which open's a browser to the clicked link.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Hyperlink_RequestNavigate(object sender, RequestNavigateEventArgs e)
{
Process.Start(new ProcessStartInfo(e.Uri.AbsoluteUri));
e.Handled = true;
}
}
}
```

AddNewFileDialog.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels.DialogViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
```

```
namespace MerlinTestStudio_Demo_Telerik.UserControls.Dialogs
{
/// <summary>
/// Interaction logic for AddNewFileDialog.xaml
/// </summary>
public partial class AddNewFileDialog : UserControl
{
public AddNewFileDialog(AddNewFileDialogVM viewModel)
{
DataContext = viewModel;

InitializeComponent();
}

private void RadButton_Click(object sender, RoutedEventArgs e)
{
CloseWindow();
}

private void NewItemListBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
if(this.NewItemListBox.SelectedItem != null)
{
(this.DataContext as AddNewFileDialogVM).AddCommand.Execute(null);
CloseWindow();
}
}

private void CloseWindow()
{
RadWindow window = this.ParentOfType<RadWindow>();
window.Close();
}

}
}
```

CloseProjectSaveDiaglog.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for CloseProjectSaveDiaglog.xaml
/// </summary>
public partial class CloseProjectSaveDiaglog : UserControl
{
public CloseProjectSaveDiaglog()
{
InitializeComponent();
}
}
}
```

ImportPinsWindowContent.xaml.cs

```
using System.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for ImportPinsWindowContent.xaml
/// </summary>
public partial class ImportPinsWindowContent : UserControl
{
public ImportPinsWindowContent()
{
InitializeComponent();
}
}
}
```

NewProjectDiaglog.xaml.cs

```
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Unity;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for NewProjectDiaglog.xaml
/// </summary>
public partial class NewProjectDiaglog : UserControl
{
public NewProjectDiaglog()
{
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<NewProjectDiaglogViewModel>()
;

InitializeComponent();
}

private void ProjectNameTextBox_GotMouseCapture(object sender, MouseEventArgs e)
{
// Fixes issue when clicking cut/copy/paste in context menu
if (ProjectNameTextBox.SelectionLength == 0)
ProjectNameTextBox.SelectAll();
}

private void ProjectNameTextBox_LostMouseCapture(object sender, MouseEventArgs e)
```

```
{
// If user highlights some text, don't override it
if (ProjectNameTextBox.SelectionLength == 0)
ProjectNameTextBox.SelectAll();

// further clicks will not select all
ProjectNameTextBox.LostMouseCapture -= ProjectNameTextBox_LostMouseCapture;
}

private void ProjectNameTextBox_LostKeyboardFocus(object sender,
KeyboardFocusChangedEventArgs e)
{
// once we've left the TextBox, return the select all behavior
ProjectNameTextBox.LostMouseCapture += ProjectNameTextBox_LostMouseCapture;
}
}
}
```

NewProjectDialog.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Dialogs
{
/// <summary>
/// Interaction logic for NewProjectDialog.xaml
/// </summary>
public partial class NewProjectDialog : UserControl
{
public NewProjectDialog()
```

```
{
InitializeComponent();
}
}
}
```

RemoveSiteWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for RemoveSiteWindow.xaml
/// </summary>
public partial class RemoveSiteWindow : UserControl
{
public RemoveSiteWindow()
{
InitializeComponent();
}

private void RadButton_Click(object sender, RoutedEventArgs e)
{
RadWindow window = this.ParentOfType<RadWindow>();
window.Close();
}
}
}
```

CalResultsExplorer.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for CalResultsExplorer.xaml
/// </summary>
public partial class CalResultsExplorer : UserControl
{
private MainWindowViewModel MVM { get { return (DataContext as PaneViewModel).MVM; } }

private List<ContextMenuItem> NullContextMenuSource => new List<ContextMenuItem>() { new
ContextMenuItem() { Text = "Import Results", IsEnabled = true }, };
private List<ContextMenuItem> CalDataModelContextMenuSource = new
List<ContextMenuItem>()
{
new ContextMenuItem() {  Text = "Create Limits File", IsEnabled = false },
new ContextMenuItem() { Text = "Remove", IsEnabled = true }
};

RadTreeViewItem lastRightClickedItem = null;
#region Constructor
public CalResultsExplorer()
{
```

```
InitializeComponent();

EventManager.RegisterClassHandler(typeof(RadTreeViewItem),
FrameworkElement.MouseRightButtonDownEvent, new
MouseButtonEventHandler(MainWindow_MouseRightButtonDown));
}
#endregion

void MainWindow_MouseRightButtonDown(object sender, MouseButtonEventArgs e)
{
if (this.lastRightClickedItem != null)
{
this.lastRightClickedItem.IsSelected = false;
}

RadTreeViewItem treeViewItem = sender as RadTreeViewItem;
treeViewItem.IsSelected = true;
treeViewItem.Focus();
this.lastRightClickedItem = treeViewItem;

e.Handled = true;
}

private void RadTreeView_ItemDoubleClick(object sender, Telerik.Windows.RadRoutedEventArgs
e)
{
//Do Nothing...
}

//Chooses which context menu items to use based on what the item is, Context menu options
should be sourced by the item being clicked on in the future.
private void TreeViewContextMenu_Opened(object sender, RoutedEventArgs e)
{
if (ResultsTree.SelectedItem != null)
{
if (ResultsTree.SelectedItem is CalResult)
{
TreeViewContextMenu.ItemsSource = null;
}
else if (ResultsTree.SelectedItem is CalDataModel)
{
#region Check if CalDataModel results contain any of the whitelisted names
```

```
foreach (var result in ((CalDataModel)ResultsTree.SelectedItem).Results)
{
if (BackendConstants.CalLimit_Producible_Results.Contains(result.ResultName))
{
CalDataModelContextMenuSource[0].IsEnabled = true; //Switch "Create Limits File" menu item
IsEnabled to true.
break;
}
else { CalDataModelContextMenuSource[0].IsEnabled = false; }
}
#endregion Check if CalDataModel results contain any of the whitelisted names

TreeViewContextMenu.ItemsSource = CalDataModelContextMenuSource;
}
}
else
{
TreeViewContextMenu.ItemsSource = NullContextMenuSource;
}
}

private void TreeViewContextMenu_ItemClick(object sender,
Telerik.Windows.RadRoutedEventArgs e)
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
switch (item.Text)
{
case "Remove":
MVM.RemoveCalResultCommand.Execute(ResultsTree.SelectedItem);
break;
case "Import Results":
MVM.ImportResultsCommand.Execute(null);
break;
case "Create Limits File":
MVM.CreateCalLimitsModelCommand.Execute(ResultsTree.SelectedItem); //Send the
corresponding Cal Data Model Results.
break;
}
}
}
}
```

ErrorList.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for ErrorList.xaml
/// </summary>
public partial class ErrorList : UserControl
{
public ErrorList()
{
InitializeComponent();
}
}
}
```

Output.xaml.cs

```
using SFP_UserControlLibrary.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for Output.xaml
/// </summary>
public partial class Output : UserControl
{
ConsoleRedirectWriter consoleRedirectWriter = new ConsoleRedirectWriter();
String LastConsoleString;

public Output()
{
InitializeComponent();
}

private void textBoxDebug_Initialized(object sender, EventArgs e)
{
// Use this for thread safe objects or UIElements in a single thread program
consoleRedirectWriter.OnWrite += delegate (string value) { LastConsoleString = value;
textBoxDebug.AppendText(value); textBoxDebug.ScrollToEnd(); };

/// Multithread operation - Use the dispatcher to write to WPF UIElements if there is more than 1
thread.
///consoleRedirectWriter.OnWrite += Dispatcher.BeginInvoke(DispatcherPriority.Normal,
(Action<string>)delegate (string value) { textBoxDebug.AppendText(value);
textBoxDebug.ScrollToEnd(); },text );
}



private void RadButton_Click(object sender, RoutedEventArgs e)
{
textBoxDebug.Text = string.Empty;
}
}
}
```

ProjectExplorer.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
```

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.TreeView;
using Telerik.Windows.DragDrop;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for ProjectExplorer.xaml
/// </summary>
public partial class ProjectExplorer : UserControl
{
private MainWindowViewModel MVM { get { return (DataContext as PaneViewModel).MVM; } }

private List<ContextMenuItem> PaneViewContextMenuSource
{
get
{
return new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = true, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
```

```
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
}
}
private List<ContextMenuItem> FolderContextMenuSource
{
get
{
return new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add New", IsEnabled = true, Command =
MVM.AddNewFileCommand }, //DataStorage.MainViewModel.AddNewFileCommand
new ContextMenuItem() { Text = "Add Existing", IsEnabled = true, Command =
MVM.AddExistingFileCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Import", IsEnabled = true, Command =
MVM.ImportAndExtractDataCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
};
}
}
private List<ContextMenuItem> SolutionContextMenuSource
{
get
{
return new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Format Test Data", IsEnabled = true, Command =
MVM.FormatProjectTestDataCommand },
//DataStorage.MainViewModel.FormatProjectTestDataCommand
new ContextMenuItem() { Text = "Properties", IsEnabled = true, Command =
MVM.ViewPropertiesCommand }, //DataStorage.MainViewModel.ViewPropertiesCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Close Project", IsEnabled = true, Command =
MVM.UnloadMerlinProjectCommand }
//DataStorage.MainViewModel.UnloadMerlinProjectCommand
};
}
```

```
}
private List<ContextMenuItem> NullContextMenuSource
{
get
{
return new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "New Project", IsEnabled = true, Command =
MVM.NewProjectFileCommand }, //DataStorage.MainViewModel.NewProjectFileCommand
new ContextMenuItem() { Text = "Open Existing Project", IsEnabled = true, Command =
MVM.AddExistingMerlinProjectCommand },
//DataStorage.MainViewModel.AddExistingMerlinProjectCommand
};
}
}

RadTreeViewItem lastRightClickedItem = null;
public ProjectExplorer()
{
InitializeComponent();

//Cancel Drop.
DragDropManager.AddDropHandler(this.ProjectTree, OnDrop, true);

//Cancel Drag Start.
DragDropManager.AddDragInitializeHandler(this.ProjectTree, OnTreeViewDragInitialize, true);

//Only disallows drop inside for PVMs, allows drop inside for folders
DragDropManager.AddDragOverHandler(ProjectTree, OnDragOver, true); //Enable only drop
inside for folders.

//Selects items upon right click.
EventManager.RegisterClassHandler(typeof(RadTreeViewItem),
FrameworkElement.MouseRightButtonDownEvent, new
MouseButtonEventHandler(MainWindow_MouseRightButtonDown));
}

#region TreeView Drag Drop Events
private void OnDrop(object sender, Telerik.Windows.DragDrop.DragEventArgs e)
{
TreeViewDragDropOptions options = DragDropPayloadManager.GetDataFromObject(e.Data,
TreeViewDragDropOptions.Key) as TreeViewDragDropOptions;
```

```
if (options != null && options.DragSourceItem != null && options.DropTargetItem != null &&
options.DropPosition != DropPosition.Undefined)
{
RadTreeViewItem dropSourceItem = options.DragSourceItem; //Item being dragged.
RadTreeViewItem dropTargetItem = options.DropTargetItem; //Target item to drop in.

if(dropSourceItem.DataContext is PVM) //Should only be PVM.
{
PVM sourcePVM = (PVM)dropSourceItem.DataContext;

if (dropTargetItem.DataContext is ProjectFolder && options.DropPosition == DropPosition.Inside )
{
ProjectFolder targetFolder = (ProjectFolder)dropTargetItem.DataContext;
//If folder section matches, drop within parent folder, other wise cancel drop. (only exeption being
waveforms)
switch (sourcePVM.ParentFolder.Section)
{
case Data.ProjectFileSection.Waveforms:
case Data.ProjectFileSection.WaveformSubFolder:
options.DropAction = targetFolder.Section == Data.ProjectFileSection.Waveforms ||
targetFolder.Section == Data.ProjectFileSection.WaveformSubFolder ? DropAction.Move :
DropAction.None;
break;
default:
options.DropAction = sourcePVM.ParentFolder.Section == targetFolder.Section ?
DropAction.Move : DropAction.None;
break;
}
}
else if (dropTargetItem.DataContext is PVM && dropTargetItem != dropSourceItem)
{
ProjectFolder targetFolder = ((PVM)dropTargetItem.DataContext).ParentFolder;
//If folder section matches, drop within parent folder, other wise cancel drop. (only exeption being
waveforms)
switch (sourcePVM.ParentFolder.Section)
{
case Data.ProjectFileSection.Waveforms:
case Data.ProjectFileSection.WaveformSubFolder:
options.DropAction = targetFolder.Section == Data.ProjectFileSection.Waveforms ||
targetFolder.Section == Data.ProjectFileSection.WaveformSubFolder ? DropAction.Move :
DropAction.None;
break;
```

```
default:
options.DropAction = sourcePVM.ParentFolder.Section == targetFolder.Section ?
DropAction.Move : DropAction.None;
break;
}
}
else
{
options.DropAction = DropAction.None;
}
}
else
{
options.DropAction = DropAction.None;
}
}
}

private void OnTreeViewDragInitialize(object sender, DragInitializeEventArgs e)
{
TreeViewDragDropOptions options = DragDropPayloadManager.GetDataFromObject(e.Data,
TreeViewDragDropOptions.Key) as TreeViewDragDropOptions;
if (options != null && options.DragSourceItem != null)
{
RadTreeViewItem draggedItem = options.DragSourceItem;
if (draggedItem.DataContext is ProjectFolder || draggedItem.DataContext is MerlinProject)
//Cancel drag start since Project and Folder are not modifiable.
{
e.Data = null;
e.DragVisual = null;
}
}
}

private void OnDragOver(object sender, Telerik.Windows.DragDrop.DragEventArgs e)
{
var options = DragDropPayloadManager.GetDataFromObject(e.Data,
TreeViewDragDropOptions.Key) as TreeViewDragDropOptions;
if (options != null && options.DropTargetItem.DataContext is PVM && options.DropPosition ==
Telerik.Windows.Controls.DropPosition.Inside)
{
options.DropPosition = DropPosition.Undefined; //places nowhere
```

```
options.UpdateDragVisual();
}
}

#endregion TreeView Drag Drop Events

void MainWindow_MouseRightButtonDown(object sender, MouseButtonEventArgs e)
{
if (this.lastRightClickedItem != null) //Left click not stored.
{
this.lastRightClickedItem.IsSelected = false;
}

RadTreeViewItem treeViewItem = sender as RadTreeViewItem;
treeViewItem.IsSelected = true;
treeViewItem.Focus();
this.lastRightClickedItem = treeViewItem;

e.Handled = true;
}

//Empty.
private void RadTreeView_ItemDoubleClick(object sender, Telerik.Windows.RadRoutedEventArgs
e)
{
//Do Nothing...
}

//Chooses which context menu items to use based on what the item is, Context menu options
should be sourced by the item being clicked on in the future.
private void RadContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ProjectTree.SelectedItems.Last() is null)
{
TreeViewContextMenu.ItemsSource = NullContextMenuSource;
return;
}

//Switch to using the .last() of the selected items, instead of selected item, to fix wrong context
menu on multi selection bug.
```

```
object lastSelectedItem = ProjectTree.SelectedItems.Last();

if (lastSelectedItem is PVM)
{
var pfs = (lastSelectedItem as PVM).ParentFolder.Section;
switch(pfs)
{
case Data.ProjectFileSection.Waveforms:
case Data.ProjectFileSection.WaveformSubFolder:
var wfMVM = (lastSelectedItem as PaneViewModel).ViewDataContext as
WaveformConverterControls.MainWindowViewModel;
if (System.IO.Path.GetExtension((lastSelectedItem as PaneViewModel).DataFilePath) == ".aiq")
{
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand },
new ContextMenuItem() { Text = "Save as .mtwf2", IsEnabled = true, Command =
MVM.SaveAsMTWFCommand},
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false }
};
}
else
{
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand },
new ContextMenuItem() { Text = "Save as .aiq", IsEnabled = true, Command =
MVM.SaveAsAIQCommand},
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false }
};
}
break;
```

```
case Data.ProjectFileSection.Product_Configurations:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = true, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Calibration_Definitions:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = true, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = true, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Connection_Manager:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
```

```
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = true, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = false, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Test_Parameters:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = false, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = false, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Test_Sequences:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = false, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = false, Command =
```

```
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Calibration_Limits:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = false, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Dll:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
```

```
case Data.ProjectFileSection.Test_Limits:
case Data.ProjectFileSection.Digital_Patterns:
case Data.ProjectFileSection.Connection_Diagrams:
TreeViewContextMenu.ItemsSource = PaneViewContextMenuSource;
break;
default:
TreeViewContextMenu.ItemsSource = null;
break;
}
}
else if (lastSelectedItem is ProjectFolder)
{
var pfs = (lastSelectedItem as ProjectFolder).Section;
switch (pfs)
{
case Data.ProjectFileSection.None:
case Data.ProjectFileSection.WaveformSubFolder:
case Data.ProjectFileSection.Connection_Manager:
case Data.ProjectFileSection.Test_Parameters:
case Data.ProjectFileSection.Test_Sequences:
TreeViewContextMenu.ItemsSource = null;
break;
case Data.ProjectFileSection.Product_Configurations:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add New", IsEnabled = true, Command =
MVM.AddNewFileCommand }, //DataStorage.MainViewModel.AddNewFileCommand
new ContextMenuItem() { Text = "Add Existing", IsEnabled = true, Command =
MVM.AddExistingFileCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
};
break;
case Data.ProjectFileSection.Dll:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Reference", IsEnabled = true, Command =
MVM.AddExistingFileCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
};
break;
case Data.ProjectFileSection.Waveforms:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Waveform", IsEnabled = true, Command =
```

```
MVM.AddExistingFileCommand }
};
break;
case Data.ProjectFileSection.Calibration_Limits:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add New", IsEnabled = false, Command =
MVM.AddNewFileCommand }, //DataStorage.MainViewModel.AddNewFileCommand
new ContextMenuItem() { Text = "Add Existing", IsEnabled = true, Command =
MVM.AddExistingFileCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Import", IsEnabled = false, Command =
MVM.ImportAndExtractDataCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
};
break;
case Data.ProjectFileSection.Connection_Diagrams:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add New", IsEnabled = true, Command =
MVM.AddNewFileCommand }, //DataStorage.MainViewModel.AddNewFileCommand
new ContextMenuItem() { Text = "Add Existing", IsEnabled = true, Command =
MVM.AddExistingFileCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
};
break;
default:
TreeViewContextMenu.ItemsSource = FolderContextMenuSource;
break;
}
}
else if (lastSelectedItem is MerlinProject)
{
TreeViewContextMenu.ItemsSource = SolutionContextMenuSource;
}
}
catch (Exception ex)
{
Console.WriteLine(ex.Message);
TreeViewContextMenu.ItemsSource = NullContextMenuSource;
}
}

private void RadContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
```

```
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
//Could pass in the Command Parameter here if dynamic, just a matter of what fires first, menu item command or this event.
switch (item.Text)
{
//case "Delete":
case "Add Reference":
case "Import":
case "Copy Full Path":
case "Add New":
case "Add Existing":
case "Add Waveform":
case "Rename":
case "Format Test Data":
case "Properties":
case "Close Project":
case "Save as .aiq":
case "Save as .mtwf2":
case "Duplicate":
item.Command.Execute(ProjectTree.SelectedItems.Last());
break;
case "Delete":
item.Command.Execute(ProjectTree.SelectedItems);
break;
case "New Project":
case "Open Existing Project":
item.Command.Execute(null);
break;
default:
//Execute no command to avoid errors.
break;
}
}
}
}
```

PropertiesPane.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for PropertiesPane.xaml
/// </summary>
public partial class PropertiesPane : UserControl
{
public PropertiesPane()
{
InitializeComponent();
}
}
}
```

SystemExplorer.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

```
namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for SystemExplorer.xaml
/// </summary>
public partial class SystemExplorer : UserControl
{
private MainWindowViewModel MVM { get { return (DataContext as PaneViewModel).MVM; } }
public SystemExplorer()
{
InitializeComponent();
}
}
}
```

DigitalLevel.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Patterns
{
/// <summary>
/// Interaction logic for DigitalLevels.xaml
/// </summary>
public partial class DigitalLevel : UserControl, IDigitalLevelsView
```

```
{
private DigitalLevelViewModel viewModelObj;
public DigitalLevel(DigitalLevelViewModel dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = DataContext as DigitalLevelViewModel;

viewModelObj.View = this;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.DigitalLevelsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.DigitalLevelsGridView);
this.PPMULevelsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.PPMULevelsGridView);
this.DCPowerLevelsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.PPMULevelsGridView);
}

//Gets fired by three different grids.
private void GridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;

if((gridView.ItemsSource as IEnumerable<object>).Count() == 0) //If Empty Add a row to each grid
if grid is empty.
{
if (gridView.ItemsSource is ObservableCollection<DigiLevel>)
{
(gridView.ItemsSource as ObservableCollection<DigiLevel>).Add(new DigiLevel());
}
else if (gridView.ItemsSource is ObservableCollection<PPMULevel>)
{
(gridView.ItemsSource as ObservableCollection<PPMULevel>).Add(new PPMULevel());
}
else if (gridView.ItemsSource is ObservableCollection<DCPowerLevel>)
{
(gridView.ItemsSource as ObservableCollection<DCPowerLevel>).Add(new DCPowerLevel());
}
```

```
}
}

private void GridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
RadGridView gridView = (RadGridView)sender;

//This Adds a new Item To the given grid view if the user edits the last row.
if (gridView.Items.IndexOf(e.Row.Item) == (gridView.Items.ItemCount - 1)) //Check to see if the
last item in which ever grid is being edited;
{
if(gridView.ItemsSource is ObservableCollection<DigiLevel>)
{
(gridView.ItemsSource as ObservableCollection<DigiLevel>).Add(new DigiLevel());
}
else if (gridView.ItemsSource is ObservableCollection<PPMULevel>)
{
(gridView.ItemsSource as ObservableCollection<PPMULevel>).Add(new PPMULevel());
}
else if (gridView.ItemsSource is ObservableCollection<DCPowerLevel>)
{
(gridView.ItemsSource as ObservableCollection<DCPowerLevel>).Add(new DCPowerLevel());
}
}
}

private void GridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
viewModelObj.CurrentLevelsData.OnChangeOccured(); //Needed since the data lists are just
strings.
}

public void ForceCommitEdit()
{
this.DigitalLevelsGridView.CommitEdit();
this.PPMULevelsGridView.CommitEdit();
this.DCPowerLevelsGridView.CommitEdit();
}
}
}
```

DigitalPattern.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
using Telerik.Windows.SyntaxEditor.Core.Text;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Patterns
{
/// <summary>
/// Interaction logic for DigitalPattern.xaml
/// </summary>
public partial class DigitalPattern : UserControl, IDigitalPatternView
{
private ObservableCollection<ContextMenuItem> ColumnContextMenuItems;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;
private DigitalPatternVM viewModelObj;

public DigitalPattern(DigitalPatternVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = DataContext as DigitalPatternVM;
```

```
viewModelObj.View = this as IDigitalPatternView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.DigitalPatternsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.DigitalPatternsGridView);

ColumnContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Register" },
new ContextMenuItem() { Text = "Rename Register" },
new ContextMenuItem() { Text = "Delete Register" }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Mode"},
new ContextMenuItem() {Text = "Remove Mode"},
new ContextMenuItem() {Text = "Duplicate Mode"},
};

}

private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
if (ClickedColumn != null) // Open parameter context menu items.
{
string header = (ClickedColumn.Content as TextBlock).Text;

foreach (string regHeader in viewModelObj.CurrentPattern.Registers)
{
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
```

```
GridContextMenu.ItemsSource = null;
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedColumn != null)
{
string header = (ClickedColumn.Content as TextBlock).Text;
int headerIndex = -1;

foreach (string regHeader in viewModelObj.CurrentPattern.Registers) //Get index of
if (header == regHeader)
headerIndex = viewModelObj.CurrentPattern.Registers.IndexOf(regHeader);

if (headerIndex != -1)
{
var regHeader = viewModelObj.CurrentPattern.Registers[headerIndex];
switch (item.Text)
{
case "Delete Register":
GridViewDataColumn gvdc = null;
foreach (var column in from GridViewDataColumn column in DigitalPatternsGridView.Columns
where column.Header.ToString() == header
select column)
{
gvdc = column;
}
if (gvdc != null)
{
DigitalPatternsGridView.Columns.Remove(gvdc);
viewModelObj.RemoveRegisterCommand.Execute(regHeader);
}
break;
case "Rename Register":
viewModelObj.RenameRegisterCommand.Execute(regHeader);
break;
case "Add Register":
viewModelObj.AddRegisterCommand.Execute(regHeader);
break;
default:
break;
```

```
}
}
}
else if (ClickedRow != null)
{
Mode mode = ClickedRow.DataContext as Mode;
switch (item.Text)
{
case "Add Mode":
viewModelObj.AddModeCommand.Execute(mode);
break;
case "Remove Mode":
viewModelObj.RemoveModeCommand.Execute(mode);
break;
case "Duplicate Mode":
//viewModelObj.DuplicateTestCommand.Execute(mode);
break;
}
}
else
{

}
}

private void GridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void GridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//RadGridView gridView = (RadGridView)sender;

//This Adds a new Item To the given grid view if the user edits the last row.
//if (gridView.Items.IndexOf(e.Row.Item) == (gridView.Items.ItemCount - 1)) //Check to see if the
last item in which ever grid is being edited;
//{
//    if (gridView.ItemsSource is ObservableCollection<Mode>)
//    {
```

```
//        (gridView.ItemsSource as ObservableCollection<Mode>).Add(new Mode());
//    }
//}
}

//Temporary Solution to getting time set name data from project tree.
private void SelectTimeSetBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
viewModelObj.ForceGetTimeSets();
}

private void DigitalPatternsGridView_CellEditEnded(object sender,
GridViewCellEditEndedEventArgs e)
{
if (e.EditAction == GridViewEditAction.Commit)
{
GridViewRowItem row = e.Cell.ParentRow;
if (row != null && row.Item is Mode)
{
Mode item = row.Item as Mode;

//Use column index to find register index and vector index since we have the parentmode.
ModeVector vector = null;
int index = DigitalPatternsGridView.Columns.IndexOf(e.Cell.Column) - GUI_Column_Count();
if(index >= 0)
{
vector = item.ModeVectors[index];
if (vector != null)
{
vector.InputValue = e.NewData as string;
viewModelObj.CurrentPattern.Vectorize_Single(vector, index);
}
}

item.FindDetectedOpcodes();
viewModelObj.CurrentPattern.Compile_Single_Mode_(item); //Rebuilt on row cell edit ended.
viewModelObj.SelectedItem = item;
}
}

viewModelObj.CurrentPattern.OnChangeOccured(); //Needed since the data lists are just strings.
}
```

```
public void AddColumn(string columnName)
{
string headerText = columnName;

int i = DigitalPatternsGridView.Columns.Count - 1; //-1 Accounts for the name column.
DigitalPatternsGridView.Columns.Add(new GridViewDataColumn()
{
Header = headerText,
DataMemberBinding = new Binding("ModeVectors[" + i + "].InputValue")
{
Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
}
});
}

public void InsertColumn(int index, string columnName)
{
string headerText = columnName;
int GUI_Index = index + 1;
DigitalPatternsGridView.Columns.Insert(GUI_Index, new GridViewDataColumn()
{
Header = headerText,
DataMemberBinding = new Binding("ModeVectors[" + index + "].InputValue")
{
Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
}
});
}

public void RemoveAtColumn(int index)
{
int GUI_Index = index + 1;
DigitalPatternsGridView.Columns.RemoveAt(GUI_Index);
}

//helper to calculate GUI column problems with static-dynamic mixtures. USE LATER.
private int GUI_Column_Count()
{
return DigitalPatternsGridView.Columns.Count - viewModelObj.CurrentPattern.Registers.Count();
```

```
}

/// <summary>
/// Clears the columns in the GridView.
/// </summary>
public void ResetTable()
{
DigitalPatternsGridView.Columns.Clear();
}

public void RebindGridView()
{
DigitalPatternsGridView.Rebind();
}

public void ForceCommitEdit()
{
this.DigitalPatternsGridView.CommitEdit();
}

//TEMP
public void ShowDPatSrcInEditor(string dpatsrcdata)
{
this.syntaxEditor.Document = new TextDocument(dpatsrcdata);
}



}
}
```

DigitalTiming.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
```

```
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Patterns
{
/// <summary>
/// Interaction logic for DigitalTiming.xaml
/// </summary>
public partial class DigitalTiming : UserControl, IDigitalTimingView
{
private DigitalTimingVM viewModelObj;

public DigitalTiming(DigitalTimingVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = DataContext as DigitalTimingVM;

viewModelObj.View = this;

this.DigiTimingGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.DigiTimingGridView);
}

private void GridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void GridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
```

```
//RadGridView gridView = (RadGridView)sender;
//
////This Adds a new Item To the given grid view if the user edits the last row.
//if (gridView.Items.IndexOf(e.Row.Item) == (gridView.Items.ItemCount - 1)) //Check to see if the last item in which ever grid is being edited;
//{
//    if (gridView.ItemsSource is ObservableCollection<DigiTimeObj>)
//    {
//        (gridView.ItemsSource as ObservableCollection<DigiTimeObj>).Add(new DigiTimeObj());
//    }
//}
}

private void DigiTimingGridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
try
{
if (e.EditAction == GridViewEditAction.Commit)
{
if (e.Cell.Column.UniqueName == "Period")
{
GridViewCellInfo cell = this.DigiTimingGridView.SelectedCells[0];
string timeSetName = (cell.Item as DigiTimeObj).Name;
double newPeriod = (double)e.NewData;

//Get list of all digitimeobjs with the same name
List<DigiTimeObj> itemsToSync = new List<DigiTimeObj>();
foreach (DigiTimeObj dto in viewModelObj.CurrentTimingData.DigiTimeSource) //LINQ extract desired.
{
if(dto.Name == timeSetName)
{
itemsToSync.Add(dto);
}
}

//Go through the newly created list and change all the "Periods" to be the same.
foreach(var dtoSync in itemsToSync)
{
dtoSync.Period = (double)e.NewData;
}
```

```
}

//viewModelObj.DataObject.OnChangeOccured();
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); Console.WriteLine(ex.Message); }

viewModelObj.CurrentTimingData.OnChangeOccured(); //Needed since the data lists are just strings.
}

public void ForceCommitEdit()
{
this.DigiTimingGridView.CommitEdit();
}
}
}
```

GenericRffePattern.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
using Telerik.Windows.Data;
using Telerik.Windows.SyntaxEditor.Core.Text;
```

```
namespace MerlinTestStudio_Demo_Telerik.UserControls.Patterns
{
/// <summary>
/// Interaction logic for DigitalPattern.xaml
/// </summary>
public partial class GenericRffePattern : UserControl, IGenericRffePatternView
{
private ObservableCollection<ContextMenuItem> ModeContextMenuItems;
private ObservableCollection<ContextMenuItem> ModeVectorContextMenuItems;
private GenericRffePatternVM viewModelObj;

public GenericRffePattern(GenericRffePatternVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = DataContext as GenericRffePatternVM;

viewModelObj.View = this as IGenericRffePatternView;

GridViewTableDefinition td = new GridViewTableDefinition() { Relation = new
PropertyRelation("ModeVectors") };
DigitalPatternsGridView.ChildTableDefinitions.Add(td);
//RowReorderBehavior.SetIsEnabled(DigitalPatternsGridView, true);

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
//this.DigitalPatternsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.DigitalPatternsGridView);

ModeContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Insert Mode", IsEnabled = true },
new ContextMenuItem() {Text = "Remove Mode", IsEnabled = true },
new ContextMenuItem() {Text = "Duplicate Mode", IsEnabled = false },
new ContextMenuItem() {IsSeparator = true },
new ContextMenuItem() {Text = "Add Data", IsEnabled = true },
};

//Icon paths not working.
ModeVectorContextMenuItems = new ObservableCollection<ContextMenuItem>()
```

```
{
//new ContextMenuItem() { IsSeparator = true },
new ContextMenuItem() {Text = "Insert Data", IsEnabled = true },
new ContextMenuItem() {Text = "Remove Data", IsEnabled = true },
};

}

//private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
//if (ClickedColumn != null) // Open parameter context menu items.
//{
//    string header = (ClickedColumn.Content as TextBlock).Text;
//
//    foreach (string regHeader in viewModelObj.CurrentPattern.Registers)
//    {
//            GridContextMenu.ItemsSource = ColumnContextMenuItems;
//    }
//}
if (ClickedRow != null) //Open test context menu items.
{
if (ClickedRow.DataContext is GenericMode)
{
GridContextMenu.ItemsSource = ModeContextMenuItems;
}
else if (ClickedRow.DataContext is GenericModeVector)
{
GridContextMenu.ItemsSource = ModeVectorContextMenuItems;
}
}
else //No context menu items.
{
GridContextMenu.ItemsSource = null;
}
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
```

```
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedRow != null)
{
if (ClickedRow.DataContext is GenericMode)
{
GenericMode mode = ClickedRow.DataContext as GenericMode;
switch (item.Text)
{
case "Insert Mode":
viewModelObj.InsertModeCommand.Execute(DigitalPatternsGridView.SelectedItems);
break;
case "Remove Mode":
viewModelObj.RemoveModeCommand.Execute(mode);
break;
case "Duplicate Mode":
//viewModelObj.DuplicateTestCommand.Execute(mode);
break;
case "Add Data":
viewModelObj.AddDataCommand.Execute(mode);
break;
}
}
else if (ClickedRow.DataContext is GenericModeVector)
{
GenericModeVector modeVector = ClickedRow.DataContext as GenericModeVector;
switch (item.Text)
{
case "Insert Data":
viewModelObj.InsertDataCommand.Execute(modeVector);
break;
case "Remove Data":
viewModelObj.RemoveDataCommand.Execute(modeVector);
break;
}
}
}
}

private void GridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
```

```
gridView.CurrentItem = null;
}

//Currently Not Used.
private void GridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//RadGridView gridView = (RadGridView)sender;

//This Adds a new Item To the given grid view if the user edits the last row.
//if (gridView.Items.IndexOf(e.Row.Item) == (gridView.Items.ItemCount - 1)) //Check to see if the
last item in which ever grid is being edited;
//{
//    if (gridView.ItemsSource is ObservableCollection<Mode>)
//    {
//        (gridView.ItemsSource as ObservableCollection<Mode>).Add(new Mode());
//    }
//}
}

//Temporary Solution to getting time set name data from project tree.
private void SelectTimeSetBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
viewModelObj.ForceGetTimeSets();
}

//Master Digital Pattern Grid View
private void DigitalPatternsGridView_CellEditEnded(object sender,
GridViewCellEditEndedEventArgs e)
{
if (e.EditAction == GridViewEditAction.Commit)
{
GridViewRowItem row = e.Cell.ParentRow;
if (row != null && row.Item is GenericMode)
{
GenericMode item = row.Item as GenericMode;

////Use column index to find register index and vector index since we have the parentmode.
//ModeVector vector = null;
//int index = DigitalPatternsGridView.Columns.IndexOf(e.Cell.Column) - GUI_Column_Count();
//if(index >= 0)
//{
```

```
//    vector = item.ModeVectors[index];
//    if (vector != null)
//    {
//        vector.InputValue = e.NewData as string;
//        viewModelObj.CurrentPattern.Vectorize_Single(vector, index, item.Spec);
//    }
//}

//TEMP for demo
//viewModelObj.CurrentPattern.Vectorize_All_For_Mode(item);

item.FindDetectedOpcodes();
viewModelObj.CurrentPattern.Compile_Single_Mode_(item); //Rebuilt on row cell edit ended.
viewModelObj.SelectedItem = item;
}

}

viewModelObj.CurrentPattern.OnChangeOccured(); //Needed since the data lists are just strings.
}

//Child Heirarchy Grid View
private void ChildGridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
if (e.EditAction == GridViewEditAction.Commit)
{
GridViewRowItem row = e.Cell.ParentRow;
//var someParent = row.Parent;
if (row != null && row.Item is GenericModeVector)
{
GenericModeVector modeVectorRow = row.Item as GenericModeVector;

viewModelObj.CurrentPattern.Vectorize_Single(modeVectorRow);

//TEMP until better way is found, possibly via datacontext of the item row hosting the child grid
view.
GenericMode parentMode = null;
foreach (GenericMode mode in viewModelObj.CurrentPattern.ModeCollection)
{
foreach (GenericModeVector mv in mode.ModeVectors)
{
if (mv == modeVectorRow)
```

```
{
parentMode = mode;
}
}
}

parentMode.FindDetectedOpcodes();
viewModelObj.CurrentPattern.Compile_Single_Mode_(parentMode); //Rebuilt on row cell edit
ended.
viewModelObj.SelectedItem = parentMode;

}
}

viewModelObj.CurrentPattern.OnChangeOccured(); //Needed since the data lists are just strings.
}

//REMOVE.
//public void AddColumn(string columnName)
//{
//    string headerText = columnName;
//
//    int i = DigitalPatternsGridView.Columns.Count - 2; //-2 Accounts for the name and spec
column.
//    DigitalPatternsGridView.Columns.Add(new GridViewDataColumn()
//    {
//        Header = headerText,
//        DataMemberBinding = new Binding("ModeVectors[" + i + "].InputValue")
//        {
//            Mode = BindingMode.TwoWay,
//            UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
//        }
//    });
//}

////REMOVE.
//public void InsertColumn(int index, string columnName)
//{
//    string headerText = columnName;
//    int GUI_Index = index + 2;
//    DigitalPatternsGridView.Columns.Insert(GUI_Index, new GridViewDataColumn()
//    {
```

```
//        Header = headerText,
//        DataMemberBinding = new Binding("ModeVectors[" + index + "].InputValue")
//        {
//            Mode = BindingMode.TwoWay,
//            UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
//        }
//    });
//}
//
//public void RemoveAtColumn(int index)
//{
//    int GUI_Index = index + 2;
//    DigitalPatternsGridView.Columns.RemoveAt(GUI_Index);
//}

////helper to calculate GUI column problems with static-dynamic mixtures. USE LATER.
//private int GUI_Column_Count()
//{
//    return DigitalPatternsGridView.Columns.Count -
(viewModelObj.CurrentPattern.Registers.Count() * 2);
//}

/// <summary>
/// Clears the columns in the GridView.
/// </summary>
public void ResetTable()
{
DigitalPatternsGridView.Columns.Clear();
}

public void RebindGridView()
{
DigitalPatternsGridView.Rebind();
}

public void ForceCommitEdit()
{
this.DigitalPatternsGridView.CommitEdit();
}

//Custom Columns (Old Code)
public void RebindDigitalPatternsGridView()
```

```
{
//try
//{
//    Style ColumnStyle = Application.Current.FindResource("TestLimits_ColumnStyle") as Style;
//Check style!!!!!!!!!!!!!!!!!!
//
//    this.DigitalPatternsGridView.ColumnGroups.Clear();
//    this.DigitalPatternsGridView.Columns.Clear();
//
//    #region Test Info Columns
//    this.DigitalPatternsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name =
"ModeInfo", Header = new TextBlock() { Text = "MODE INFO", VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } });
//    //Add the Test Info columns
//    this.DigitalPatternsGridView.Columns.Add(new GridViewDataColumn()
//    {
//        UniqueName = "Name",
//        Header = "Name",
//        DataMemberBinding = new Binding("Name"),
//        ColumnGroupName = "ModeInfo",
//        Style = ColumnStyle
//    });
//    this.DigitalPatternsGridView.Columns.Add(new GridViewComboBoxColumn()
//    {
//        UniqueName = "Spec",
//        Header = "Spec",
//        DataMemberBinding = new Binding("Spec"),
//        DisplayMemberPath = "Value",
//        SelectedValueMemberPath = "Key",
//        ItemsSource = (this.DataContext as DigitalPatternVM).PatternSpecKvPs,
//        ColumnGroupName = "ModeInfo",
//        EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick
//    });
//    #endregion Test Info Columns
//
//    #region Hard Bins and Multi Bin Columns
//    var hardBinGroup = new GridViewColumnGroup() { Name = "Registers", Header = new
TextBlock() { Text = "REGISTERS", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } };
//    this.DigitalPatternsGridView.ColumnGroups.Add(hardBinGroup);
//
//    #region Generate Multi Pass Columns
```

```
//    if ((this.DataContext as DigitalPatternVM).CurrentPattern.ModeCollection.Count != 0)
//    {
//        (this.DataContext as DigitalPatternVM).RegisterHeaders.Clear(); //Reset column group
name tracking
//        for (int i = 0; i < (this.DataContext as DigitalPatternVM).CurrentPattern.Registers.Count; i++)
//        {
//            string regName = (this.DataContext as DigitalPatternVM).CurrentPattern.Registers[i];
//            hardBinGroup.ChildGroups.Add(new GridViewColumnGroup() { Name = regName,
Header = new TextBlock() { Text = regName, VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
//            (this.DataContext as DigitalPatternVM).RegisterHeaders.Add(regName); //Add Column
group name track
//
//            this.DigitalPatternsGridView.Columns.Add(new GridViewDataColumn()
//            {
//                UniqueName = "SA " + (i + 1),
//                Header = "SA",
//                DataMemberBinding = new Binding("ModeVectors[" + i + "].UserIDValue"),
//                ColumnGroupName = regName,
//                Style = ColumnStyle
//            });
//            this.DigitalPatternsGridView.Columns.Add(new GridViewDataColumn()
//            {
//                UniqueName = "Data " + (i + 1),
//                Header = "Data",
//                DataMemberBinding = new Binding("ModeVectors[" + i + "].InputValue"),
//                ColumnGroupName = regName,
//                Style = ColumnStyle
//            });
//
//        }
//    }
//    #endregion Generate Multi Pass Columns
//
//    #endregion Hard Bins and Multi Bin Columns
//
//}
//catch (Exception ex) { MessageBox.Show(ex.Message); }
}

//TEMP
public void ShowDPatSrcInEditor(string dpatsrcdata)
```

```
{
this.syntaxEditor.Document = new TextDocument(dpatsrcdata);
}



}
}
```

ConditionsData.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.Runtime.Remoting.Messaging;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
using Telerik.Windows.DragDrop;
using UnitConversionLib;
using Unity;
using System.Linq;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestConditions
{
/// <summary>
/// Interaction logic for ConditionsData.xaml
/// </summary>
public partial class ConditionsData : UserControl, IConditionsView
{
private ObservableCollection<MenuItem> UnitContextMenuItems;
private ObservableCollection<MenuItem> TestRowContextMenuItems;
private ConditionsViewModel viewModelObj;

public ConditionsData(object dataContext)
{
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<ConditionsViewModel>();
```

```
InitializeComponent();

this.DataContext = dataContext;

viewModelObj = (DataContext as ConditionsViewModel);

viewModelObj.View = this as IConditionsView;

////Auto Generates the columns manually to avoid duplicates.
//foreach (Data.ParameterMapTag pmt in viewModelObj.PVM.ParentProject.ParameterMapTags)
//    AddColumn(pmt.ColumnName, pmt.Unit);

RowReorderBehavior.SetIsEnabled(ConditionsGrid, true);
DragDropManager.AddDragDropCompletedHandler(ConditionsGrid, new
DragDropCompletedEventHandler(OnDragDropCompleted));

UnitContextMenuItems = new ObservableCollection<MenuItem>()
{
new MenuItem()
{
Text = "Change Unit",
SubItems = new ObservableCollection<MenuItem>()
{
new MenuItem() {
Text = "Volts",
SubItems = new ObservableCollection<MenuItem>()
{
new MenuItem() { Text = "V" },
new MenuItem() { Text = "dV" },
new MenuItem() { Text = "cV" },
new MenuItem() { Text = "mV" },
new MenuItem() { Text = "uV" },
new MenuItem() { Text = "nV" },
new MenuItem() { Text = "pV" }
}
},
new MenuItem()
{
Text = "Amps",
SubItems = new ObservableCollection<MenuItem>()
{
```

```
new MenuItem() { Text = "A" },
new MenuItem() { Text = "dA" },
new MenuItem() { Text = "cA" },
new MenuItem() { Text = "mA" },
new MenuItem() { Text = "uA" },
new MenuItem() { Text = "nA" },
new MenuItem() { Text = "pA" }
}
},
new MenuItem()
{
Text = "Frequency",
SubItems = new ObservableCollection<MenuItem>()
{
new MenuItem() { Text = "THz" },
new MenuItem() { Text = "GHz" },
new MenuItem() { Text = "MHz" },
new MenuItem() { Text = "KHz" },
new MenuItem() { Text = "hHz" },
new MenuItem() { Text = "daHz" },
new MenuItem() { Text = "Hz" }
}
},
new MenuItem()
{
Text = "Power",
SubItems = new ObservableCollection<MenuItem>()
{
new MenuItem() { Text = "dBm" },
new MenuItem() { Text = "dB" }
}
},
new MenuItem() { Text = "Number" },
new MenuItem() { Text = "Percentage" }
}
},
new MenuItem() { Text = "Delete Test Parameter" }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<MenuItem>()
{
```

```
new MenuItem() {Text = "Insert New", IconPath = "/Resources/Images/MenuItems/AddIcon.png"},
new MenuItem() {Text = "Remove", IconPath =
"/Resources/Images/MenuItems/RemoveIcon.png"},
new MenuItem() {Text = "Duplicate", IconPath = "/Resources/Images/MenuItems/Clone.png"},
new MenuItem() {Text = "Test Numeration Break", IconPath = ""}
};
}

private void OnDragDropCompleted(object sender, DragDropCompletedEventArgs e)
{
viewModelObj.RenumerateTestsCommand.Execute(null);
}

private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }



private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedColumn != null) // Open parameter context menu items.
{
string header = (ClickedColumn.Content as TextBlock).Text;

foreach (ParameterMapTag pmt in viewModelObj.ParentProject.ParameterMapTags)
if (header == pmt.ColumnName || header == $"{pmt.ColumnName} - {pmt.Unit}")
GridContextMenu.ItemsSource = !pmt.IsUnitable ? null : UnitContextMenuItems;
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

//Fires when the unit of a parameter has been changed.
private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
```

```
try
{
MenuItem item = (e.OriginalSource as RadMenuItem).DataContext as MenuItem;
if (ClickedColumn != null)
{
string header = (ClickedColumn.Content as TextBlock).Text;
int headerIndex = -1;

foreach (ParameterMapTag pmt in viewModelObj.ParentProject.ParameterMapTags)
if (header == pmt.ColumnName || header == $"{pmt.ColumnName} - {pmt.Unit}")
headerIndex = pmt.ColumnIndex;

if (headerIndex != -1)
{
var pmt = viewModelObj.ParentProject.ParameterMapTags[headerIndex];
switch (item.Text)
{
case "Delete Test Parameter":
GridViewDataColumn gvdc = null;
foreach (var column in from GridViewDataColumn column in ConditionsGrid.Columns
where column.Header.ToString() == header
select column)
{
gvdc = column;
}
if (gvdc != null)
{
ConditionsGrid.Columns.Remove(gvdc);
viewModelObj.DeleteTestParameterCommand.Execute(pmt);
}
break;
default:
string oldUnit = pmt.Unit;
string newUnit = item.Text;

pmt.Unit = newUnit;
string newHeader = pmt.ColumnName + " - " + newUnit;
(ClickedColumn.Content as TextBlock).Text = newHeader;

bool IsOfSameUnitType = false;
foreach (string unit in pmt.Units)
if (pmt.Unit == unit)
```

```
IsOfSameUnitType = true;

if (IsOfSameUnitType)
{
MessageBoxResult messageBoxResult = System.Windows.MessageBox.Show("Would you like to
convert all values?", "Conversion Confirmation", System.Windows.MessageBoxButton.YesNo);
if (messageBoxResult == MessageBoxResult.Yes)
foreach (ITest t in viewModelObj.ParentProject.Tests)
try
{
string modifiedValue;
Data.Services.UnitService.ValueUnitModifier(t.Parameters[pmt.ColumnIndex].Value.ToString(),
oldUnit, newUnit, out modifiedValue);
t.Parameters[pmt.ColumnIndex].Value = modifiedValue;
}
catch (Exception ex) { }
}
else
{
string unitType;
Data.Services.UnitService.MatchUnit(newUnit, out unitType, out _);
pmt.UnitType = unitType;
}
break;
}
}
}
else if (ClickedRow != null)
{
ITest testRow = ClickedRow.DataContext as ITest;
switch (item.Text)
{
case "Insert New":
viewModelObj.InsertTestCommand.Execute(testRow);
break;
case "Remove":
viewModelObj.RemoveTestCommand.Execute(testRow);
break;
case "Duplicate":
viewModelObj.DuplicateTestCommand.Execute(testRow);
break;
case "Test Numeration Break":
```

```
viewModelObj.DesignateTestNumerationBreakCommand.Execute(testRow);
break;
}
}
else
{

}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

private void LocationInput_KeyUp(object sender, System.Windows.Input.KeyEventArgs e)
{
if (ConditionsGrid.SelectedItem != null)
ConditionsGrid.BringIndexIntoView((ConditionsGrid.ItemsSource as
ObservableCollection<Test>).IndexOf(ConditionsGrid.SelectedItem as Test));
}



/// <summary>
/// Adds the column to the GridView.
/// </summary>
public void AddColumn(string columnName, string unit)
{
try
{
string headerText = string.IsNullOrEmpty(unit) ? columnName : $"{columnName} - {unit}";

int i = ConditionsGrid.Columns.Count;
ConditionsGrid.Columns.Add(new GridViewDataColumn()
{
Header = headerText,
DataMemberBinding = new Binding("Parameters[" + i + "].Value")
{
Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
}
});
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
```

```
/// <summary>
/// Clears the columns in the GridView.
/// </summary>
public void ResetTable()
{
ConditionsGrid.Columns.Clear();
}

public void RebindGridView()
{
ConditionsGrid.Rebind();
}

private void UserControl_Unloaded(object sender, System.Windows.RoutedEventArgs e)
{
//Method needs to be updated/rebuilt.
//DataManagement.GroupAttributeCheck(); //Makes sure groups still match (either on tab close or project close?).
//viewModelObj = null;
}

public void ForceCommitEdit()
{
this.ConditionsGrid.CommitEdit();
}

private void ConditionsGrid_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void ConditionsGrid_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
(this.DataContext as ConditionsViewModel).OnChangeOccured();
}
}

#region ContextMenu MenuItemObjects

public class MenuItem : INotifyPropertyChanged
```

```
{
private bool isEnabled = true;
private string text;
private string iconPath;
private ObservableCollection<MenuItem> subItems;

public bool IsEnabled
{
get
{
return this.isEnabled;
}
set
{
if (this.isEnabled != value)
{
this.isEnabled = value;
this.OnNotifyPropertyChanged("IsEnabled");
}
}
}
public string Text
{
get
{
return this.text;
}
set
{
if (this.text != value)
{
this.text = value;
this.OnNotifyPropertyChanged("Text");
}
}
}
public string IconPath
{
get
{
return this.iconPath;
}
```

```
set
{
if (this.iconPath != value)
{
this.iconPath = value;
this.OnNotifyPropertyChanged("IconPath");
}
}
}
public ObservableCollection<MenuItem> SubItems
{
get
{
if (this.subItems == null)
{
this.subItems = new ObservableCollection<MenuItem>();
}
return this.subItems;
}
set
{
if (this.subItems != value)
{
this.subItems = value;
this.OnNotifyPropertyChanged("SubItems");
}
}
}

#region INPC Members

public event PropertyChangedEventHandler PropertyChanged;
private void OnNotifyPropertyChanged(string ptopertyName)
{
if (this.PropertyChanged != null)
{
this.PropertyChanged(this, new PropertyChangedEventArgs(ptopertyName));
}
}

#endregion
}
```

```
#endregion
}
```

GoldLimits.xaml.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestLimits
{
/// <summary>
/// Interaction logic for GoldLimits.xaml
/// </summary>
public partial class GoldLimits : UserControl, IGoldLimitsView
{
public GoldLimitsVM viewModelObj;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;

public GoldLimits(GoldLimitsVM dataContext)
{
InitializeComponent();

this.DataContext = dataContext;
```

```
viewModelObj = dataContext;

viewModelObj.View = this as IGoldLimitsView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.GoldLimitsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.GoldLimitsGridView);

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Test", IsEnabled = false, Command =
DataStorage.MainViewModel.AddTestToActiveProjectCommand },
new ContextMenuItem() {Text = "Remove Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.RemoveTestFromActiveProjectCommand},
new ContextMenuItem() {Text = "Duplicate Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.DuplicateTestToActiveProjectCommand},
};
}

private CommonColumnHeader ClickedColumnGroup { get { return
GridContextMenu.GetClickedElement<CommonColumnHeader>(); } }
private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private void GoldLimitsGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void GoldLimitsGridView_CellEditEnded(object sender,
Telerik.Windows.Controls.GridViewCellEditEndedEventArgs e)
{
viewModelObj.OnChangeOccured();
}

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
```

```
try
{
if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedRow != null)
{
GoldLimit limit = ClickedRow.DataContext as GoldLimit;
switch (item.Text)
{
case "Add Test":
case "Remove Test":
case "Duplicate Test":
item.Command.Execute(null);
break;
}
}
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

public void ForceCommitEdit()
{
this.GoldLimitsGridView.CommitEdit();
}
}
}
```

OffsetLimits.xaml.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
```

```
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestLimits
{
/// <summary>
/// Interaction logic for OffsetLimits.xaml
/// </summary>
public partial class OffsetLimits : UserControl , IOffsetLimitsView
{
private OffsetLimitsVM viewModelObj;
private ObservableCollection<ContextMenuItem> ColumnContextMenuItems;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;

public OffsetLimits(OffsetLimitsVM dataContext)
{
InitializeComponent();

this.DataContext = dataContext;

viewModelObj = dataContext;

viewModelObj.View = this as IOffsetLimitsView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.OffsetLimitsGridView.KeyboardCommandProvider = new
```

```
CustomKeyboardCommandProvider(this.OffsetLimitsGridView);

ColumnContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Site", IsEnabled = true, Command =
viewModelObj.AddOffsetSiteCommand },
new ContextMenuItem() { Text = "Delete Site", IsEnabled = true, Command =
viewModelObj.RemoveOffsetSiteCommand }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Test", IsEnabled = false, Command =
DataStorage.MainViewModel.AddTestToActiveProjectCommand },
new ContextMenuItem() {Text = "Remove Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.RemoveTestFromActiveProjectCommand},
new ContextMenuItem() {Text = "Duplicate Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.DuplicateTestToActiveProjectCommand},
};
}

private CommonColumnHeader ClickedColumnGroup { get { return
GridContextMenu.GetClickedElement<CommonColumnHeader>(); } }
private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private void OffsetLimitsGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void OffsetLimitsGridView_CellEditEnded(object sender,
Telerik.Windows.Controls.GridViewCellEditEndedEventArgs e)
{
viewModelObj.OnChangeOccured();
}

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
```

```
{
try
{
if (ClickedColumn != null) //if the framework element is not null.
{
string header = (ClickedColumn.Content as TextBlock).Text;
if (header.Contains("Site"))
{
ColumnContextMenuItems[1].IsEnabled = true;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
else
{
ColumnContextMenuItems[1].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedColumn != null)
{
string header = (ClickedColumn.Content as TextBlock).Text;
int headerIndex = -1;
if (header.Contains("Site"))
{
headerIndex = viewModelObj.OffsetSiteHeaders.FindIndex(x => x == header);
}

switch (item.Text)
{
case "Delete Site":
```

```
if (headerIndex != -1)
{
item.Command.Execute(headerIndex);
}
break;
case "Add Site":
item.Command.Execute(null);
break;
default:
break;
}
}
else if (ClickedRow != null)
{
OffsetLimit limit = ClickedRow.DataContext as OffsetLimit;
switch (item.Text)
{
case "Add Test":
case "Remove Test":
case "Duplicate Test":
item.Command.Execute(null);
break;
}
}
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

public void RebindGridView()
{
this.OffsetLimitsGridView.Rebind();
}

public void ForceCommitEdit()
{
this.OffsetLimitsGridView.CommitEdit();
}

public void RebindOffsetLimitsGridView()
{
try
{
```

```
Style ColumnStyle = Application.Current.FindResource("TestLimits_ColumnStyle") as Style;
Style editorStyle = Application.Current.FindResource("DropDownOnCellEdit") as Style;

this.OffsetLimitsGridView.ColumnGroups.Clear();
this.OffsetLimitsGridView.Columns.Clear();

#region Test Info Columns
this.OffsetLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "TestInfo",
Header = new System.Windows.Controls.TextBlock() { Text = "TEST INFO", VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } });
//Add the Test Info columns
this.OffsetLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestNumber",
Header = "Test Number",
DataMemberBinding = new Binding("TestNumber"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.OffsetLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestName",
Header = "Test Name",
TextAlignment = TextAlignment.Left,
DataMemberBinding = new Binding("TestName"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.OffsetLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "Units",
Header = "Units",
DataMemberBinding = new Binding("Units"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
#endregion Test Info Columns

#region Offset Columns
var hardBinGroup = new GridViewColumnGroup() { Name = "Offsets", Header = new
System.Windows.Controls.TextBlock() { Text = "OFFSETS", VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } };
```

```
this.OffsetLimitsGridView.ColumnGroups.Add(hardBinGroup);

this.OffsetLimitsGridView.Columns.Add(new GridViewComboBoxColumn()
{
UniqueName = "ApplyOffset",
Header = "Apply",
DataMemberBinding = new Binding("ApplyOffset"),
ItemsSource = Enum.GetValues(typeof(OffsetControl)).Cast<OffsetControl>(),
ColumnGroupName = "Offsets",
Style = ColumnStyle,
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick,
EditorStyle = editorStyle,
MinWidth = 100
}); ;

#region Generate Offset Site Columns
if (viewModelObj.OffsetLimitsObj.Data.Count != 0)
{
viewModelObj.OffsetSiteHeaders.Clear(); //Reset column group name tracking
for (int i = 0; i < viewModelObj.MaxOffsetSitesCount; i++)
{
viewModelObj.OffsetSiteHeaders.Add(("Site " + (i + 1))); //Add Column group name track
this.OffsetLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "OffsetSite " + (i + 1),
Header = ("Site " + (i + 1)),
DataMemberBinding = new Binding("OffsetSites[" + i + "]"),
ColumnGroupName = "Offsets",
Style = ColumnStyle
});
}
}
#endregion Generate Multi Pass Columns

#endregion
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
}
}
```

TestFixtures.xaml.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestLimits
{
/// <summary>
/// Interaction logic for TestFixtures.xaml
/// </summary>
public partial class TestFixtures : UserControl, ITestFixturesView
{
private TestFixturesVM viewModelObj;
private ObservableCollection<ContextMenuItem> ColumnContextMenuItems;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;

public TestFixtures(TestFixturesVM dataContext)
{
InitializeComponent();

this.DataContext = dataContext;

viewModelObj = dataContext;

viewModelObj.View = this as ITestFixturesView;
```

```
//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.TestFixturesGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.TestFixturesGridView);

ColumnContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Site", IsEnabled = true, Command =
viewModelObj.AddTestFixtureSiteCommand },
new ContextMenuItem() { Text = "Delete Site", IsEnabled = true, Command =
viewModelObj.RemoveTestFixtureSiteCommand }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Test", IsEnabled = false, Command =
DataStorage.MainViewModel.AddTestToActiveProjectCommand },
new ContextMenuItem() {Text = "Remove Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.RemoveTestFromActiveProjectCommand},
new ContextMenuItem() {Text = "Duplicate Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.DuplicateTestToActiveProjectCommand},
};
}

private CommonColumnHeader ClickedColumnGroup { get { return
GridContextMenu.GetClickedElement<CommonColumnHeader>(); } }
private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private void TestFixturesGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void TestFixturesGridView_CellEditEnded(object sender,
Telerik.Windows.Controls.GridViewCellEditEndedEventArgs e)
{
viewModelObj.OnChangeOccured();
```

```
}

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedColumn != null) //if the framework element is not null.
{
string header = (ClickedColumn.Content as TextBlock).Text;
if (header.Contains("Site"))
{
ColumnContextMenuItems[1].IsEnabled = true;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
else
{
ColumnContextMenuItems[1].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedColumn != null)
{
string header = (ClickedColumn.Content as TextBlock).Text;
int headerIndex = -1;
if (header.Contains("Site"))
{
headerIndex = viewModelObj.TestFixtureSiteHeaders.FindIndex(x => x == header);
}
```

```
switch (item.Text)
{
case "Delete Site":
if (headerIndex != -1)
{
item.Command.Execute(headerIndex);
}
break;
case "Add Site":
item.Command.Execute(null);
break;
default:
break;
}
}
else if (ClickedRow != null)
{
TraceLossLimit limit = ClickedRow.DataContext as TraceLossLimit;
switch (item.Text)
{
case "Add Test":
case "Remove Test":
case "Duplicate Test":
item.Command.Execute(null);
break;
}
}
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

public void ForceCommitEdit()
{
this.TestFixturesGridView.CommitEdit();
}

public void RebindGridView()
{
this.TestFixturesGridView.Rebind();
}

public void RebindOffsetLimitsGridView()
```

```
{
try
{
Style ColumnStyle = Application.Current.FindResource("TestLimits_ColumnStyle") as Style;
Style editorStyle = Application.Current.FindResource("DropDownOnCellEdit") as Style;

this.TestFixturesGridView.ColumnGroups.Clear();
this.TestFixturesGridView.Columns.Clear();

#region Test Info Columns
this.TestFixturesGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "TestInfo",
Header = new TextBlock() { Text = "TEST INFO", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
//Add the Test Info columns
this.TestFixturesGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestNumber",
Header = "Test Number",
DataMemberBinding = new Binding("TestNumber"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TestFixturesGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestName",
Header = "Test Name",
TextAlignment = TextAlignment.Left,
DataMemberBinding = new Binding("TestName"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TestFixturesGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "Units",
Header = "Units",
DataMemberBinding = new Binding("Units"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
#endregion Test Info Columns

#region Offset Columns
```

```
var hardBinGroup = new GridViewColumnGroup() { Name = "TestFixtures", Header = new
TextBlock() { Text = "TEST FIXTURES", VerticalAlignment =  VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } };
this.TestFixturesGridView.ColumnGroups.Add(hardBinGroup);

this.TestFixturesGridView.Columns.Add(new GridViewComboBoxColumn()
{
UniqueName = "ApplyTestFixture",
Header = "Apply",
DataMemberBinding = new Binding("ApplyTestFixture"),
ItemsSource = Enum.GetValues(typeof(OffsetControl)).Cast<OffsetControl>(),
ColumnGroupName = "TestFixtures",
Style = ColumnStyle,
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick,
EditorStyle = editorStyle,
MinWidth = 100
}); ;

#region Generate Offset Site Columns
if (viewModelObj.TestFixturesObj.Data.Count != 0)
{
viewModelObj.TestFixtureSiteHeaders.Clear(); //Reset column group name tracking
for (int i = 0; i < viewModelObj.MaxTestFixtureSitesCount; i++)
{
viewModelObj.TestFixtureSiteHeaders.Add(("Site " + (i + 1))); //Add Column group name track
this.TestFixturesGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestFixtureSite " + (i + 1),
Header = ("Site " + (i + 1)),
DataMemberBinding = new Binding("TestFixtureSites[" + i + "]"),
ColumnGroupName = "TestFixtures",
Style = ColumnStyle
});
}
}
#endregion Generate Multi Pass Columns

#endregion
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
}
```

}

TestLimits.xaml.cs

```
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestLimits
{
/// <summary>
/// Interaction logic for TestLimits.xaml
/// </summary>
public partial class TestLimits : UserControl, ITestLimitsView, IDisposable
{
private TestLimitsVM viewModelObj;
private ObservableCollection<ContextMenuItem> ColumnContextMenuItems;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;

public TestLimits(TestLimitsVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = dataContext;
```

```
((TestLimitsVM)this.DataContext).View = this;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.TestLimitsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.TestLimitsGridView);

ColumnContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Multi-Pass", IsEnabled = true, Command =
viewModelObj.AddMultiPassBinCommand },
new ContextMenuItem() { Text = "Add Multi-Fail", IsEnabled = true, Command =
viewModelObj.AddMultiFailBinCommand },
new ContextMenuItem() { Text = "Delete Multi-Pass", IsEnabled = true, Command =
viewModelObj.RemoveMultiPassBinCommand },
new ContextMenuItem() { Text = "Delete Multi-Fail", IsEnabled = true, Command =
viewModelObj.RemoveMultiFailBinCommand }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Test", IsEnabled = false, Command =
DataStorage.MainViewModel.AddTestToActiveProjectCommand },
new ContextMenuItem() {Text = "Remove Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.RemoveTestFromActiveProjectCommand},
new ContextMenuItem() {Text = "Duplicate Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.DuplicateTestToActiveProjectCommand},
};
}
~TestLimits()
{
//Console.WriteLine("Test Limits Control Finalized..."); //Casuses non owner thread error which
then causes memory leak fatal error on exit. //Need custom console messenger.
}

private CommonColumnHeader ClickedColumnGroup { get { return
GridContextMenu.GetClickedElement<CommonColumnHeader>(); } }
private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }
```

```
private void TestLimitsGridView_CellEditEnded(object sender,
Telerik.Windows.Controls.GridViewCellEditEndedEventArgs e)
{
(this.DataContext as TestLimitsVM).OnChangeOccured(); //May be needed since Multi Bin objetcs
don't yet have INPC.
}

//Needs better sensing to know when to display what options.
private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedColumnGroup != null) //if the framework element is not null.
{
string header = (string)ClickedColumnGroup.FindChildByType<TextBlock>().Text;
if (header.Contains("MULTI"))
{
if (header.Contains("PASS"))
{
ColumnContextMenuItems[2].IsEnabled = true;
ColumnContextMenuItems[3].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
else if (header.Contains("FAIL"))
{
ColumnContextMenuItems[2].IsEnabled = false;
ColumnContextMenuItems[3].IsEnabled = true;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else
{
ColumnContextMenuItems[2].IsEnabled = false;
ColumnContextMenuItems[3].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else if (ClickedColumn != null)
```

```
{
//Incase column group headers disappear...
ColumnContextMenuItems[2].IsEnabled = false;
ColumnContextMenuItems[3].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch(Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedColumnGroup != null)
{
string header = (string)ClickedColumnGroup.FindChildByType<TextBlock>().Text;
int headerIndex = -1;
if (header.Contains("PASS"))
{
headerIndex = (this.DataContext as TestLimitsVM).MultiPassHeaders.FindIndex(x => x ==
header);
//foreach (string regHeader in (this.DataContext as TestLimitsVM).MultiPassHeaders) //Get index
of
//    if (header == regHeader)
//        headerIndex = (this.DataContext as TestLimitsVM).MultiPassHeaders.IndexOf(regHeader);
}
else if (header.Contains("FAIL"))
{
headerIndex = (this.DataContext as TestLimitsVM).MultiFailHeaders.FindIndex(x => x == header);
//foreach (string regHeader in (this.DataContext as TestLimitsVM).MultiFailHeaders) //Get index of
//    if (header == regHeader)
//        headerIndex = (this.DataContext as TestLimitsVM).MultiFailHeaders.IndexOf(regHeader);
}

switch (item.Text)
{
case "Delete Multi-Pass":
case "Delete Multi-Fail":
```

```
if (headerIndex != -1)
{
item.Command.Execute(headerIndex);
//(this.DataContext as TestLimitsVM).RemoveMultiPassBinCommand.Execute(headerIndex);
}
break;
case "Add Multi-Pass":
case "Add Multi-Fail":
item.Command.Execute(null);
//(this.DataContext as TestLimitsVM).AddMultiPassBinCommand.Execute(null);
break;
default:
break;
}

if (headerIndex != -1)
{

}
}
else if (ClickedRow != null)
{
UpperLowerLimit limit = ClickedRow.DataContext as UpperLowerLimit;
switch (item.Text)
{
case "Add Test":
case "Remove Test":
case "Duplicate Test":
item.Command.Execute(null);
break;
}
}
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void TestLimitsGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}
```

```
public void RebindGridView()
{
this.TestLimitsGridView.Rebind();
}

public void ForceCommitEdit()
{
this.TestLimitsGridView.CommitEdit();
}

public void RebindTestLimitsGridView()
{
try
{
Style ColumnStyle = Application.Current.FindResource("TestLimits_ColumnStyle") as Style;

this.TestLimitsGridView.ColumnGroups.Clear();
this.TestLimitsGridView.Columns.Clear();

#region Test Info Columns
this.TestLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "TestInfo",
Header = new TextBlock() { Text = "TEST INFO", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
//Add the Test Info columns
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestNumber",
Header = "Test Number",
DataMemberBinding = new Binding("TestNumber"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestName",
Header = "Test Name",
TextAlignment = TextAlignment.Left,
DataMemberBinding = new Binding("TestName"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
```

```
{
UniqueName = "Units",
Header = "Units",
DataMemberBinding = new Binding("Units"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
#endregion Test Info Columns

#region FT QA upper lower Columns
//Add the FT-QA upper/lower columns
this.TestLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "FT", Header
= new TextBlock() { Text = "FT", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
this.TestLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "QA", Header
= new TextBlock() { Text = "QA", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "FTLower",
Header = "Lower",
DataMemberBinding = new Binding("FTLower"),
ColumnGroupName = "FT",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "FTUpper",
Header = "Upper",
DataMemberBinding = new Binding("FTUpper"),
ColumnGroupName = "FT",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "QALower",
Header = "Lower",
DataMemberBinding = new Binding("QALower"),
ColumnGroupName = "QA",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
```

```
{
UniqueName = "QAUpper",
Header = "Upper",
DataMemberBinding = new Binding("QAUpper"),
ColumnGroupName = "QA",
Style = ColumnStyle
});
#endregion FT QA upper lower Columns

#region Hard Bins and Multi Bin Columns
var hardBinGroup = new GridViewColumnGroup() { Name = "HardBin", Header = new TextBlock()
{ Text = "HARD BIN", VerticalAlignment = VerticalAlignment.Center, HorizontalAlignment =
HorizontalAlignment.Center } };
hardBinGroup.ChildGroups.Add(new GridViewColumnGroup() { Name = "Fail", Header = new
TextBlock() { Text = "FAIL", VerticalAlignment = VerticalAlignment.Center, HorizontalAlignment =
HorizontalAlignment.Center } });
this.TestLimitsGridView.ColumnGroups.Add(hardBinGroup);

#region Static Hard Bin Columns
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "HardBinNumber",
Header = "Number",
DataMemberBinding = new Binding("HardBinNumber"),
ColumnGroupName = "Fail",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "HardBinName",
Header = "Hard Bin Name",
DataMemberBinding = new Binding("HardBinName"),
ColumnGroupName = "Fail",
Style = ColumnStyle
});
//this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
//{
//    UniqueName = "HardBinPF",
//    Header = "Flag",
//    DataMemberBinding = new Binding("HardBinPF"),
//    ColumnGroupName = "Fail",
//    Style = ColumnStyle
```

```
//});
#endregion Static Hard Bin Columns

#region Generate Multi Pass Columns
if ((this.DataContext as TestLimitsVM).TestLimitsObj.Data.Count != 0)
{
(this.DataContext as TestLimitsVM).MultiPassHeaders.Clear(); //Reset column group name
tracking
for (int i = 0; i < (this.DataContext as TestLimitsVM).MaxMultiPassBinsCount; i++)
{
hardBinGroup.ChildGroups.Add(new GridViewColumnGroup() { Name = ("Multi Pass " + (i + 1)),
Header = new TextBlock() { Text = ("MULTI PASS " + (i + 1)), VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } });
(this.DataContext as TestLimitsVM).MultiPassHeaders.Add(("MULTI PASS " + (i + 1))); //Add
Column group name track
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiPassHardBinNumber " + (i + 1),
Header = "Number",
DataMemberBinding = new Binding("MultiPassBins[" + i + "].HardBinNumber"),
ColumnGroupName = ("Multi Pass " + (i + 1)),
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiPassHardBinName " + (i + 1),
Header = "Name",
DataMemberBinding = new Binding("MultiPassBins[" + i + "].HardBinName"),
ColumnGroupName = ("Multi Pass " + (i + 1)),
Style = ColumnStyle
});
//this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
//{
//    UniqueName = "MultiPassHardBinPF " + (i + 1),
//    Header = "Flag",
//    DataMemberBinding = new Binding("MultiPassBins[" + i + "].HardBinPF"),
//    ColumnGroupName = ("Multi Pass " + (i + 1)),
//    Style = ColumnStyle
//});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiPassLower " + (i + 1),
```

```
Header = "Lower",
DataMemberBinding = new Binding("MultiPassBins[" + i + "].Lower"),
ColumnGroupName = ("Multi Pass " + (i + 1)),
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiPassUpper " + (i + 1),
Header = "Upper",
DataMemberBinding = new Binding("MultiPassBins[" + i + "].Upper"),
ColumnGroupName = ("Multi Pass " + (i + 1)),
Style = ColumnStyle
});
}
}
#endregion Generate Multi Pass Columns

#region Generate Multi Fail Columns
if ((this.DataContext as TestLimitsVM).TestLimitsObj.Data.Count != 0)
{
(this.DataContext as TestLimitsVM).MultiFailHeaders.Clear();
for (int i = 0; i < (this.DataContext as TestLimitsVM).MaxMultiFailBinsCount; i++)
{
hardBinGroup.ChildGroups.Add(new GridViewColumnGroup() { Name = ("Multi Fail " + (i + 1)),
Header = new TextBlock() { Text = ("MULTI FAIL " + (i + 1)), VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center }  });
(this.DataContext as TestLimitsVM).MultiFailHeaders.Add(("MULTI FAIL " + (i + 1)));
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiFailSoftBinNumber " + (i + 1),
Header = "Number",
DataMemberBinding = new Binding("MultiFailBins[" + i + "].SoftBinNumber"),
ColumnGroupName = ("Multi Fail " + (i + 1)),
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiFailSoftBinName " + (i + 1),
Header = "Name",
DataMemberBinding = new Binding("MultiFailBins[" + i + "].SoftBinName"),
ColumnGroupName = ("Multi Fail " + (i + 1)),
Style = ColumnStyle
```

```
});
//this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
//{
//    UniqueName = "MultiFailSoftBinPF " + (i + 1),
//    Header = "Flag",
//    DataMemberBinding = new Binding("MultiFailBins[" + i + "].SoftBinPF"),
//    ColumnGroupName = ("Multi Fail " + (i + 1)),
//    Style = ColumnStyle
//});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiSoftBinLower " + (i + 1),
Header = "Lower",
DataMemberBinding = new Binding("MultiFailBins[" + i + "].Lower"),
ColumnGroupName = ("Multi Fail " + (i + 1)),
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiSoftBinUpper " + (i + 1),
Header = "Upper",
DataMemberBinding = new Binding("MultiFailBins[" + i + "].Upper"),
ColumnGroupName = ("Multi Fail " + (i + 1)),
Style = ColumnStyle
});
}
}
#endregion

#endregion Hard Bins and Multi Bin Columns

#region Soft Bin Columns
//Add the Soft Bin Columns
this.TestLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "SoftBin",
Header = new TextBlock() { Text = "SOFT BIN", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "SoftBinNumber",
Header = "Number",
DataMemberBinding = new Binding("SoftBinNumber"),
ColumnGroupName = "SoftBin",
```

```
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "SoftBinName",
Header = "Soft Bin Name",
DataMemberBinding = new Binding("SoftBinName"),
ColumnGroupName = "SoftBin",
Style = ColumnStyle
});
//this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
//{
//    UniqueName = "SoftBinPF",
//    Header = "Flag",
//    DataMemberBinding = new Binding("SoftBinPF"),
//    ColumnGroupName = "SoftBin",
//    Style = ColumnStyle
//});
#endregion Soft Bin Columns
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
//this.DataContext = viewModelObj;
//viewModelObj.View = this as ITestLimitsView;
}

private void UserControl_Unloaded(object sender, RoutedEventArgs e)
{
//this.DataContext = null;
//viewModelObj.View = null;
//GC.Collect();
}

public void Dispose()
{
this.viewModelObj = null;
this.DataContext = null;
this.ColumnContextMenuItems = null;
this.TestRowContextMenuItems = null;
```

```
this.Content = null;
}
}
}
```

TracelossLimits.xaml.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestLimits
{
/// <summary>
/// Interaction logic for TracelossLimits.xaml
/// </summary>
public partial class TracelossLimits : UserControl, ITracelossLimitsView
{
private TracelossLimitsVM viewModelObj;
private ObservableCollection<ContextMenuItem> ColumnContextMenuItems;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;

public TracelossLimits(TracelossLimitsVM dataContext)
{
InitializeComponent();
```

```
this.DataContext = dataContext;

viewModelObj = dataContext;

viewModelObj.View = this as ITracelossLimitsView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.TracelossLimitsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.TracelossLimitsGridView);

ColumnContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Site", IsEnabled = true, Command =
viewModelObj.AddTracelossSiteCommand },
new ContextMenuItem() { Text = "Delete Site", IsEnabled = true, Command =
viewModelObj.RemoveTracelossSiteCommand }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Test", IsEnabled = false, Command =
DataStorage.MainViewModel.AddTestToActiveProjectCommand },
new ContextMenuItem() {Text = "Remove Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.RemoveTestFromActiveProjectCommand},
new ContextMenuItem() {Text = "Duplicate Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.DuplicateTestToActiveProjectCommand},
};
}

private CommonColumnHeader ClickedColumnGroup { get { return
GridContextMenu.GetClickedElement<CommonColumnHeader>(); } }
private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
```

```
if (ClickedColumn != null) //if the framework element is not null.
{
string header = (ClickedColumn.Content as TextBlock).Text;
if (header.Contains("Site"))
{
ColumnContextMenuItems[1].IsEnabled = true;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
else
{
ColumnContextMenuItems[1].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedColumn != null)
{
string header = (ClickedColumn.Content as TextBlock).Text;
int headerIndex = -1;
if (header.Contains("Site"))
{
headerIndex = viewModelObj.TracelossSiteHeaders.FindIndex(x => x == header);
}

switch (item.Text)
{
case "Delete Site":
if (headerIndex != -1)
{
item.Command.Execute(headerIndex);
```

```
}
break;
case "Add Site":
item.Command.Execute(null);
break;
default:
break;
}
}
else if (ClickedRow != null)
{
TraceLossLimit limit = ClickedRow.DataContext as TraceLossLimit;
switch (item.Text)
{
case "Add Test":
case "Remove Test":
case "Duplicate Test":
item.Command.Execute(null);
break;
}
}
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void TracelossLimitsGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void TracelossLimitsGridView_CellEditEnded(object sender,
Telerik.Windows.Controls.GridViewCellEditEndedEventArgs e)
{
viewModelObj.OnChangeOccured();
}

public void ForceCommitEdit()
{
this.TracelossLimitsGridView.CommitEdit();
}
```

```
public void RebindGridView()
{
this.TracelossLimitsGridView.Rebind();
}

public void RebindOffsetLimitsGridView()
{
try
{
Style ColumnStyle = Application.Current.FindResource("TestLimits_ColumnStyle") as Style;
Style editorStyle = Application.Current.FindResource("DropDownOnCellEdit") as Style;

this.TracelossLimitsGridView.ColumnGroups.Clear();
this.TracelossLimitsGridView.Columns.Clear();

#region Test Info Columns
this.TracelossLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name =
"TestInfo", Header = new System.Windows.Controls.TextBlock() { Text = "TEST INFO",
VerticalAlignment = VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center }
});
//Add the Test Info columns
this.TracelossLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestNumber",
Header = "Test Number",
DataMemberBinding = new Binding("TestNumber"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TracelossLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestName",
Header = "Test Name",
TextAlignment = TextAlignment.Left,
DataMemberBinding = new Binding("TestName"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TracelossLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "Units",
Header = "Units",
```

```
DataMemberBinding = new Binding("Units"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
#endregion Test Info Columns

#region Offset Columns
var hardBinGroup = new GridViewColumnGroup() { Name = "Traceloss", Header = new
System.Windows.Controls.TextBlock() { Text = "TRACELOSS", VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } };
this.TracelossLimitsGridView.ColumnGroups.Add(hardBinGroup);

this.TracelossLimitsGridView.Columns.Add(new GridViewComboBoxColumn()
{
UniqueName = "ApplyTraceloss",
Header = "Apply",
DataMemberBinding = new Binding("ApplyTraceLoss"),
ItemsSource = Enum.GetValues(typeof(OffsetControl)).Cast<OffsetControl>(),
ColumnGroupName = "Traceloss",
Style = ColumnStyle,
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick,
EditorStyle = editorStyle,
MinWidth = 100
}); ;

#region Generate Offset Site Columns
if (viewModelObj.TracelossLimitsObj.Data.Count != 0)
{
viewModelObj.TracelossSiteHeaders.Clear(); //Reset column group name tracking
for (int i = 0; i < viewModelObj.MaxTracelossSitesCount; i++)
{
viewModelObj.TracelossSiteHeaders.Add(("Site " + (i + 1))); //Add Column group name track
this.TracelossLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TracelossSite " + (i + 1),
Header = ("Site " + (i + 1)),
DataMemberBinding = new Binding("TracelossSites[" + i + "]"),
ColumnGroupName = "Traceloss",
Style = ColumnStyle
});
}
}
```

```
#endregion Generate Multi Pass Columns

#endregion
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

}
}
```

DUT_End_Test.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestSequences
{
/// <summary>
/// Interaction logic for DUT_End_Test.xaml
/// </summary>
public partial class DUT_End_Test : UserControl
{
public DUT_End_Test()
{
InitializeComponent();
DataContext = new ViewModels.DUT_End_TestViewModel();
}
}
}
```

DUT_Start_Test.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestSequences
{
/// <summary>
/// Interaction logic for DUT_Start_Test.xaml
/// </summary>
public partial class DUT_Start_Test : UserControl
{
public DUT_Start_Test()
{
InitializeComponent();
DataContext = new ViewModels.DUT_Start_TestViewModel();
}
}
}
```

DUT_Test.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using Telerik.Windows.Controls;
using Telerik.Windows.Data;
using System.Collections.ObjectModel;
using System.ComponentModel;
using Telerik.Windows.Controls.GridView;
using Unity;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Windows.Input;
```

```
using MerlinTestStudio_Demo_Telerik.Data;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestSequences
{
/// <summary>
/// Interaction logic for DUT_Test.xaml
/// </summary>
public partial class DUT_Test : UserControl, IDUTTestView
{
private ObservableCollection<MenuItem> rowContextMenuItems;

private DUT_TestViewModel viewModelObj;


public DUT_Test(DUT_TestViewModel dataContext)
{

InitializeComponent();

DataContext = dataContext;
//((UnityContainer)Application.Current.Resources["IoC"]).Resolve<DUT_TestViewModel>();

viewModelObj = (DataContext as DUT_TestViewModel);

viewModelObj.View = this as IDUTTestView;


GridViewTableDefinition td = new GridViewTableDefinition() { Relation = new
PropertyRelation("Tests") };
HierarchicalGridView.ChildTableDefinitions.Add(td);
RowReorderBehavior.SetIsEnabled(HierarchicalGridView, true);
HierarchicalGridView.Drop += DropChanges;
FunctionSequenceListDUT.PreviewDrop += DropChanges; //Custom DragDropBehavior
presenting challenges will fix soon.
FunctionsLibraryList.PreviewDrop += DropChanges;

ObservableCollection<MenuItem> groupItems = new ObservableCollection<MenuItem>()
{
new MenuItem() { Text = "Add to Group" },
new MenuItem() { Text = "Group Matching" },
new MenuItem() { Text = "Break Group" },
new MenuItem() { Text = "Skip Remaining" }
```

```
};
this.rowContextMenuItems = groupItems;
}

private void DropChanges(object sender, DragEventArgs e)
{
viewModelObj.ForceOnChangesMade();
}

#region Private Methods

/// <summary>
/// This is called when ever a(n) ComboBox selection is changed in the
GridViewComboBoxColumns.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void OnSelectionChanged(object sender, SelectionChangedEventArgs e)
{
if (e.AddedItems.Count > 0)
{
RadComboBox comboBox = e.OriginalSource as RadComboBox;
if (comboBox.SelectedValue != null)
{
string value = comboBox.SelectedValue.ToString();
if(e.RemovedItems.Count > 0)
{
foreach(string s in viewModelObj.CommandStrings)
{
if (e.RemovedItems[0].ToString() == s)
{
viewModelObj.PreviouslySelectedCommand = s;
}
}
}
viewModelObj.SelectedInComboBoxChangedCommand.Execute(value);
}
}
}

private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
```

```
//Handler added once UC_loaded so data added in before doesn't trigger event for each
ComboBox
HierarchicalGridView.AddHandler(RadComboBox.SelectionChangedEvent, new
System.Windows.Controls.SelectionChangedEventHandler(OnSelectionChanged));
}

private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

#endregion

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedRow != null)
{
HierarchicalGridView.SelectedItem = ClickedRow.DataContext;
var selected = (ISequenceItem)HierarchicalGridView.SelectedItem;
foreach (var item in rowContextMenuItems)
if (selected == null) return;
if (selected is IGroupable == true)
{
rowContextMenuItems[0].IsEnabled = false;
rowContextMenuItems[1].IsEnabled = false;
rowContextMenuItems[2].IsEnabled = true;
rowContextMenuItems[3].IsEnabled = true;
GridContextMenu.ItemsSource = rowContextMenuItems;
}
else if (selected is IGroupable == false)
{
rowContextMenuItems[0].IsEnabled = true;
rowContextMenuItems[1].IsEnabled = true;
rowContextMenuItems[2].IsEnabled = false;
rowContextMenuItems[3].IsEnabled = true;
GridContextMenu.ItemsSource = rowContextMenuItems;
}
else if (selected is SequenceBlock == true)
{
rowContextMenuItems[0].IsEnabled = false;
rowContextMenuItems[1].IsEnabled = false;
rowContextMenuItems[2].IsEnabled = false;
```

```
rowContextMenuItems[3].IsEnabled = true;
GridContextMenu.ItemsSource = rowContextMenuItems;
}
}
else
{
foreach (var item in this.rowContextMenuItems)
{
item.IsEnabled = false;
}
GridContextMenu.ItemsSource = rowContextMenuItems;
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
MenuItem item = (e.OriginalSource as RadMenuItem).DataContext as MenuItem;
ISequenceItem SeqItemRow = ClickedRow.DataContext as ISequenceItem;
switch (item.Text)
{
case "Add to Group":
viewModelObj.AddTestToGroupCommand.Execute(SeqItemRow);
break;
case "Group Matching":
viewModelObj.GroupMatchingCommand.Execute(SeqItemRow);
break;
case "Break Group":
viewModelObj.BreakGroupCommand.Execute(SeqItemRow);
break;
case "Skip Remaining":
viewModelObj.SkipRemainingCommand.Execute(SeqItemRow);
break;
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

private void LocationInput_KeyUp(object sender, System.Windows.Input.KeyEventArgs e)
```

```
{
if(HierarchicalGridView.SelectedItem != null)
HierarchicalGridView.BringIndexIntoView((HierarchicalGridView.ItemsSource as
ObservableCollection<ISequenceItem>).IndexOf(HierarchicalGridView.SelectedItem as
ISequenceItem));
}

#region IDUTTestView implementation

public void ClearGridViewFocus()
{
try
{
var scope = System.Windows.Input.FocusManager.GetFocusScope(HierarchicalGridView);
System.Windows.Input.FocusManager.SetFocusedElement(scope, null);
System.Windows.Input.Keyboard.ClearFocus();
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

public void ScrollObjectIntoView(ISequenceItem obj)
{
this.HierarchicalGridView.ScrollIntoView(obj);
}

public void RebindGridView()
{
HierarchicalGridView.Rebind();
}

#endregion

private void FunctionSequenceListDUT_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
FunctionsLibraryList.SelectedItem = null;
}

private void ValMapToggleButton_Checked(object sender, RoutedEventArgs e)
{

}
```

```
public void ForceCommitEdit()
{
this.HierarchicalGridView.CommitEdit();
}

private void HierarchicalGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void HierarchicalGridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs
e)
{
(this.DataContext as DUT_TestViewModel).OnChangeOccured();
}
}

#region ContextMenu MenuItemObjects

public class MenuItem : INotifyPropertyChanged
{
private bool isEnabled = true;
private string text;
private ObservableCollection<MenuItem> subItems;

public bool IsEnabled
{
get
{
return this.isEnabled;
}
set
{
if (this.isEnabled != value)
{
this.isEnabled = value;
this.OnNotifyPropertyChanged("IsEnabled");
}
}
}
```

```
public string Text
{
get
{
return this.text;
}
set
{
if (this.text != value)
{
this.text = value;
this.OnNotifyPropertyChanged("Text");
}
}
}
public ObservableCollection<MenuItem> SubItems
{
get
{
if (this.subItems == null)
{
this.subItems = new ObservableCollection<MenuItem>();
}
return this.subItems;
}
set
{
if (this.subItems != value)
{
this.subItems = value;
this.OnNotifyPropertyChanged("SubItems");
}
}
}

#region INPC Members

public event PropertyChangedEventHandler PropertyChanged;
private void OnNotifyPropertyChanged(string ptopertyName)
{
if (this.PropertyChanged != null)
{
```

```
this.PropertyChanged(this, new PropertyChangedEventArgs(ptopertyName));
}
}

#endregion
}

#endregion

}
```

SessionLoad.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestSequences
{
/// <summary>
/// Interaction logic for SessionLoad.xaml
/// </summary>
public partial class Session_Load : UserControl
{
public Session_Load()
{
InitializeComponent();
DataContext = new ViewModels.SessionLoadViewModel();
}
}
}
```

SessionUnload.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestSequences
{
/// <summary>
/// Interaction logic for SessionUnload.xaml
/// </summary>
public partial class SessionUnload : UserControl
{
public SessionUnload()
{
InitializeComponent();
DataContext = new ViewModels.SessionUnloadViewModel();
}
}
}
```

AutoGeneratedChecker.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Tools
{
/// <summary>
/// Interaction logic for AutoGeneratedChecker.xaml
/// </summary>
public partial class AutoGeneratedChecker : UserControl
{
public AutoGeneratedChecker()
{
InitializeComponent();
}
}
}
```

DataFormattingTool.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Unity;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Tools
{
/// <summary>
/// Interaction logic for DataFormattingTool.xaml
/// </summary>
public partial class DataFormattingTool : UserControl
```

```
{
public DataFormattingTool(DataFormattingToolViewModel dataContext)
{
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<DataFormattingToolViewModel>(
);
InitializeComponent();

this.DataContext = dataContext;
}
}
}
```

FunctionSequencingTool.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Services;
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Linq;
using Telerik.Windows.Controls;
using MerlinTestStudio_Demo_Telerik.Data.Models;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Tools
{
/// <summary>
/// Interaction logic for FunctionSequencingTool.xaml
/// </summary>
public partial class FunctionSequencingTool : UserControl
{
MerlinProject PassedProject { get; set; }

public FunctionSequencingTool(MerlinProject project)
{
InitializeComponent();
PassedProject = project;
SourceComboBox.ItemsSource = project.DLLs;
}

public SequenceCollection<Data.Function> PassedSequenceCollection
{
get { return
```

```
(SequenceCollection<Data.Function>)this.GetValue(PassedSequenceCollectionProperty); }
set { this.SetValue(PassedSequenceCollectionProperty, value); }
}

public static readonly DependencyProperty PassedSequenceCollectionProperty =
DependencyProperty.Register(
"PassedSequenceCollection", typeof(SequenceCollection<Data.ISequenceItemComponent>),
typeof(FunctionSequencingTool), new PropertyMetadata(new
SequenceCollection<Data.ISequenceItemComponent>()));

private void ImportBtn_Click(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
// Create OpenFileDialog
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog() { Title = "Import
Extension", DefaultExt = ".dll", Filter = "Application Extension (.dll)|*.dll",  };

bool? result = dlg.ShowDialog();

if (result == true)
{
var dll_Types = MerlinProject.ImportAssmblyTypes(dlg.FileName);
foreach (Data.DLL dll in dll_Types)
PassedProject.DLLs.Add(dll);

if (dll_Types.Count > 0)
SourceComboBox.SelectedItem = dll_Types[0]; //Selects the first item of a newly imported group
of dll types.
}
}

private void CancelBtn_Click(object sender, RoutedEventArgs e)
{
PassedSequenceCollection.Clear(); //Empty the collection.

RadWindow window = this.ParentOfType<RadWindow>();
window.Close();
}
}
}
```

GeneratePinsFromCalibrationDefinition.xaml.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Tools
{
/// <summary>
/// Interaction logic for GeneratePinsFromCalibrationDefinition.xaml
/// </summary>
public partial class GeneratePinsFromCalibrationDefinition : UserControl
{
public GeneratePinsFromCalibrationDefinition()
{
InitializeComponent();
}
}
}
```

TestStudioGridView.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Tools
{
//IDEA CLASS possibly for future use.
public class TestStudioGridView : RadGridView
{
```

```
public TestStudioGridView()
{
// Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.KeyboardCommandProvider = new CustomKeyboardCommandProvider(this);
}




}
}
```

DataFormattingToolViewModel.cs

```
using System;
using Telerik.Windows.Documents.Spreadsheet.Model;
using Telerik.Windows.Controls;
using System.Windows.Media;
using System.Windows.Input;
using Telerik.Windows.Controls.Spreadsheet.Worksheets;
using System.Data;
using System.Collections.ObjectModel;
using MerlinTestStudio_Demo_Telerik.Data;
using System.Linq;
using MerlinTestStudio_Demo_Telerik.Data.Services;
using System.Windows;
using MT.TestStudio.GUI.ViewModels;
using System.Collections.Generic;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using System.IO;
using System.Text.RegularExpressions;
using MerlinTestStudio_Demo_Telerik.Data.Models;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public class DataFormattingToolViewModel : PVM //, IPVMLink
{
//public PaneViewModel PVM
//{
//    get;
//    set;
//}
```

```
#region Private Data Definitions
private Color yellow = Color.FromRgb(255, 255, 153); //Light yellow = #FFFF99
private Color orange = Color.FromRgb(255, 204, 153); //Light orange = #FFCC99
private Color red = Color.FromRgb(255, 153, 153); //Light red = #FF9999
private Color green = Color.FromRgb(204, 255, 153); //Light green = #CCFF99
private Color blue_green = Color.FromRgb(153, 255, 204); //Light blue-green = #99FFCC
private Color cyan = Color.FromRgb(153, 255, 255); //Light cyan = #99FFFF
private Color blue = Color.FromRgb(153, 204, 255); //Light blue = #99CCFF
private Color blue_purple = Color.FromRgb(153, 153, 255); //Light blue-purple = #9999FF
private Color purple = Color.FromRgb(204, 153, 255); //Light purple = #CC99FF
private Color pink = Color.FromRgb(255, 153, 255); //Light pink = #FF99FF
private Color grey_default = Color.FromRgb(224, 224, 224); //Light grey as a defualt color

private static Enum[] tagTypeArray = new Enum[] //Neccessary Tags
{
TestAttributeType.Test_Name, //Test_Number
TestAttributeType.Test_Number, //Test_Name
TestAttributeType.Test_Unit, //Test_Unit
TestAttributeType.Function_Call, //Function_Call
TestAttributeType.Group, // Group
///TagType.Pin_Parameter,
///TagType.Pin_Parameter,
///TagType.Pin,
///TagType.Pout,
///TagType.Pout_Tolerance,
///TagType.Frequency,
///TagType.Waveform
};

private int DataBegin = 0;
private int DataEnding = 0;
private int NumOfTests = 0;
#endregion

#region Private Members
readonly List<string> _unitTypes = new List<string>() { "Volts", "Amps", "Frequency", "Power",
"Percentage", "Number" };
private ObservableCollection<DataTag> tags = new ObservableCollection<DataTag>();
private Workbook dataWorkbook = new Workbook();
private Selection selected;
private DataTag _selectedTag;
#endregion
```

```
#region Public Members
public List<string> unitsTypes { get { return _unitTypes; } }

public ObservableCollection<DataTag> Tags
{
get { return tags; }
set { tags = value; OnPropertyChanged("Tags"); }
}

public ObservableCollection<Worksheet> ImportedWorkBooks
{
get; set;
}

public Workbook DataWorkBook
{
get { return dataWorkbook; }
set
{
if (dataWorkbook != value)
{
dataWorkbook = value;

//Clears all fill color properties in each sheet when imported.
foreach (Worksheet ws in dataWorkbook.Sheets)
{
ws.Cells[ws.UsedCellRange].SetFill(new PatternFill(PatternType.Solid, Colors.Transparent,
Colors.Transparent));
}

OnPropertyChanged("DataWorkBook");
}
}
}

public Selection Selected
{
get { return selected; }
set
{
selected = value;
```

```
CellRange baseRange = selected.ActiveRange.SelectedCellRange;
selected.Select(new CellRange(baseRange.FromIndex.RowIndex,
baseRange.FromIndex.ColumnIndex, baseRange.FromIndex.RowIndex,
baseRange.ToIndex.ColumnIndex), true);
OnPropertyChanged("Selected");
}
}

public DataTag SelectedTag
{
get { return _selectedTag; }
set { _selectedTag = value; OnPropertyChanged("SelectedTag"); }
}
#endregion

#region Constructor
public override void PaneClose()
{
//this.View = null; //No View to nullify.
}

public event Action<string> OnChangesMade = (UserControl) => { };
public DataFormattingToolViewModel(Type contentType, MerlinProject parentProject) :
base(contentType, parentProject)
{
OnChangesMade += (UserControl) =>
{
//_viewModelLocator.OnControlInteract(UserControl);
IsSaveRequired = true;
};

}
#endregion

#region Commands

#region Selection Changed Command
public ICommand SelectionChangedCommand { get { return new
DelegateCommand(SelectionChanged); } }
//Sets the selection on MouseLeftButtonUp
private void SelectionChanged(object obj) //Work around??
{
```

```
Selection selection = obj as Selection;
Selected = selection;
}
#endregion

#region Reset Tests Method and Numerate Tests Command
private void ResetTests()
{
NumOfTests = 0;
DataEnding = 0;
DataBegin = 0;
Tags = new ObservableCollection<DataTag>();
}

public ICommand NumerateTestsCommand { get { return new
DelegateCommand(NumerateTests); } }
private void NumerateTests(object obj)
{
try
{
if
(Selected.ActiveCell.GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat)
!= "")
{
ResetTests();
CellIndex StartIndex = Selected.ActiveCellIndex;
int rowIndex = Selected.ActiveCellIndex.RowIndex;
int columnIndex = Selected.ActiveCellIndex.ColumnIndex;

int attempts = 5; //Gives the while loop 5 tries to find the first test.
while (string.IsNullOrEmpty(DataWorkBook.ActiveWorksheet.Cells[rowIndex + 1,
columnIndex].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat)))
{
rowIndex++;
attempts--;
if (attempts <= 0) { Selected.Select(StartIndex, true); return; }
}
DataBegin = rowIndex + 1;
while (DataWorkBook.ActiveWorksheet.Cells[rowIndex + 1,
columnIndex].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat) != "")
{
rowIndex++;
```

```
NumOfTests++;
}
DataEnding = rowIndex;
NumOfTests = DataEnding - DataBegin;

//Adds the Test Number Tag manually.
Tags.Add(new DataTag(false) { ColumnName = "Test Number", ColumnIndex = columnIndex,
AttributeType = TestAttributeType.Test_Number });

//Sets the Fill of the cells for visualization of the tagged data.
Selected.Cells.SetFill(new PatternFill(PatternType.Solid, yellow, Colors.Transparent));
DataWorkBook.ActiveWorksheet.Cells[new CellRange(DataBegin, StartIndex.ColumnIndex,
rowIndex, columnIndex)].SetFill(new PatternFill(PatternType.Solid, yellow, Colors.Transparent));
}
}
catch (Exception ex)
{
System.Windows.MessageBox.Show(ex.Message);
}
}
#endregion

#region Numerate Pattern Modes
public ICommand NumeratePatternModesCommand { get { return new
DelegateCommand(NumeratePatternModes); } }
private void NumeratePatternModes(object obj)
{
try
{
if
(Selected.ActiveCell.GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat)
!= "")
{
ResetTests();
CellIndex StartIndex = Selected.ActiveCellIndex;
int rowIndex = Selected.ActiveCellIndex.RowIndex;
int columnIndex = Selected.ActiveCellIndex.ColumnIndex;

int attempts = 5; //Gives the while loop 5 tries to find the first test.
while (string.IsNullOrEmpty(DataWorkBook.ActiveWorksheet.Cells[rowIndex + 1,
columnIndex].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat)))
{
```

```
rowIndex++;
attempts--;
if (attempts <= 0) { Selected.Select(StartIndex, true); return; }
}
DataBegin = rowIndex + 1;
while (DataWorkBook.ActiveWorksheet.Cells[rowIndex + 1,
columnIndex].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat) != "")
{
rowIndex++;
NumOfTests++;
}
DataEnding = rowIndex;
NumOfTests = DataEnding - DataBegin;

//Adds the Test Number Tag manually.
Tags.Add(new DataTag(false) { ColumnName = "Name", ColumnIndex = columnIndex,
AttributeType = TestAttributeType.Test_Number });

//Sets the Fill of the cells for visualization of the tagged data.
Selected.Cells.SetFill(new PatternFill(PatternType.Solid, purple, Colors.Transparent));
DataWorkBook.ActiveWorksheet.Cells[new CellRange(DataBegin, StartIndex.ColumnIndex,
rowIndex, columnIndex)].SetFill(new PatternFill(PatternType.Solid, purple, Colors.Transparent));
}
}
catch (Exception ex)
{
System.Windows.MessageBox.Show(ex.Message);
}
}
#endregion

#region Tag Test Parameter Data
//Test Parameter Tagging.
public ICommand TagDataCommand { get { return new DelegateCommand(TagDataMethod); } }
private void TagDataMethod(object obj)
{
if (DataEnding <= 0) { System.Windows.MessageBox.Show("Please Tag Test Numbers before
tagging any other data."); return; }

Color color = new Color();
TestAttributeType tagType;
```

```
string tag = obj as string;
switch (tag)
{
case "Test Name": color = yellow; tagType = TestAttributeType.Test_Name; break;
//case "Test Number": color = yellow; tagType = TagType.Test_Number; break;
case "Test Unit": color = yellow; tagType = TestAttributeType.Test_Unit; break;
case "Group": color = green; tagType = TestAttributeType.Group; break;
case "Function Call": color = green; tagType = TestAttributeType.Function_Call; break;
case "Pin Parameter": color = orange; tagType = TestAttributeType.Pin_Parameter; break;
//case "RF_Source": color = blue; tagType = TagType.RF_Source; break;
//case "RF_Meas": color = blue; tagType = TagType.RF_Meas; break;
case "Pin": color = blue; tagType = TestAttributeType.Pin; break;
case "Pout": color = blue; tagType = TestAttributeType.Pout; break;
case "Pout_Tolerance": color = blue; tagType = TestAttributeType.Pout_Tolerance; break;
case "Frequency": color = blue; tagType = TestAttributeType.Frequency; break;
case "Waveform": color = blue; tagType = TestAttributeType.Waveform; break;
case "Digital Mode": color = blue_purple; tagType = TestAttributeType.Digital_Mode; break;
case "Parameter": color = orange; tagType = TestAttributeType.Parameter; break;
//case "Parameter Unit": color = red; tagType = TagType.Parameter_Unit; break;

//case "PatternMode": color = purple; tagType = TestAttributeType.Pattern_Mode_Name; break;
case "Register": color = purple; tagType = TestAttributeType.Pattern_Mode_Register; break;

default: color = grey_default; tagType = TestAttributeType.Default; break;
}

try
{
if
(Selected.ActiveCell.GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat)
!= "")
{
CellIndex fromIndex = Selected.ActiveRange.SelectedCellRange.FromIndex;
CellIndex toIndex = Selected.ActiveRange.SelectedCellRange.ToIndex;
var TagList = Tags.ToList();

for (int c = fromIndex.ColumnIndex; c <= toIndex.ColumnIndex; c++) // Iterates through the
selected column range.
{
int index = TagList.FindIndex(item => item.TagCellIndex == new CellIndex(fromIndex.RowIndex,
c));
if (index < 0)
```

```
{
string columnName = DataWorkBook.ActiveWorksheet.Cells[fromIndex.RowIndex,
c].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat); //Gets the
selected column name.
string above = DataWorkBook.ActiveWorksheet.Cells[fromIndex.RowIndex - 1,
c].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat).Replace(" ", "");
string below = DataWorkBook.ActiveWorksheet.Cells[fromIndex.RowIndex + 1,
c].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat).Replace(" ", "");

var newTag = new DataTag(false) { ColumnName = columnName, ColumnIndex = c, RowIndex =
fromIndex.RowIndex, AttributeType = tagType };

string UnitType = ""; string Unit = "";
if (!string.IsNullOrEmpty(above))
Data.Services.UnitService.MatchUnit(above, out UnitType, out Unit);
if (!string.IsNullOrEmpty(below))
Data.Services.UnitService.MatchUnit(below, out UnitType, out Unit);

if (newTag.IsUnitable)
{
newTag.UnitType = UnitType;
newTag.Unit = Unit;
}

//Only run this check if the user is tagging RFFEPAT registers.
//Checks if the data only contains only C-style hex notation (0xFF) values
if (tagType == TestAttributeType.Pattern_Mode_Register &&
!DigitalPatternModel.OnlyHexInString(columnName))
{
//RemoveRegisterAndWhitespace(columnName);

MessageBox.Show($"Cannot tag [{columnName}] due to invalid register value, please ensure tag
value is in C-style hex notation '0xFF'.");
continue; //Skips the remaining code for this tag addition loop.
}

Tags.Add(newTag); //Add tthe tag to the global unprocessed tag collection.

Selected.Cells.SetFill(new PatternFill(PatternType.Solid, color, Colors.Transparent));
DataWorkBook.ActiveWorksheet.Cells[new CellRange(DataBegin, c, DataEnding, c)].SetFill(new
PatternFill(PatternType.Solid, color, Colors.Transparent));
```

```
}
}
}
}
catch (Exception ex)
{
System.Windows.MessageBox.Show(ex.Message);
}
}
#endregion

/// <summary>
/// Removes the word "register", any  whitespace, and underscore characters from the input string.
/// </summary>
/// <param name="input"></param>
/// <returns></returns>
public static string RemoveRegisterAndWhitespace(string input)
{
string output = Regex.Replace(input, @"\s", "");
output = Regex.Replace(output, @"register", "", RegexOptions.IgnoreCase);
output = Regex.Replace(output, @"_", "");
return output;
}

#region Tag Test Limit Data
//Test Parameter Tagging.
public ICommand TagTestLimitDataCommand { get { return new
DelegateCommand(TagDataMethod); } }
private void TagTestLimitDataMethod(object obj)
{
if (DataEnding <= 0) { System.Windows.MessageBox.Show("Please Tag Test Numbers before
tagging any other data."); return; }

Color color = new Color();
TestAttributeType tagType;

string tag = obj as string;
switch (tag)
{
case "Test Name": color = yellow; tagType = TestAttributeType.Test_Name; break;
//case "Test Number": color = yellow; tagType = TagType.Test_Number; break;
case "Test Unit": color = yellow; tagType = TestAttributeType.Test_Unit; break;
```

```
case "Group": color = green; tagType = TestAttributeType.Group; break;
case "Function Call": color = green; tagType = TestAttributeType.Function_Call; break;
case "Pin Parameter": color = orange; tagType = TestAttributeType.Pin_Parameter; break;
//case "RF_Source": color = blue; tagType = TagType.RF_Source; break;
//case "RF_Meas": color = blue; tagType = TagType.RF_Meas; break;
case "Pin": color = blue; tagType = TestAttributeType.Pin; break;
case "Pout": color = blue; tagType = TestAttributeType.Pout; break;
case "Pout_Tolerance": color = blue; tagType = TestAttributeType.Pout_Tolerance; break;
case "Frequency": color = blue; tagType = TestAttributeType.Frequency; break;
case "Waveform": color = blue; tagType = TestAttributeType.Waveform; break;
case "Digital Mode": color = blue_purple; tagType = TestAttributeType.Digital_Mode; break;
case "Parameter": color = orange; tagType = TestAttributeType.Parameter; break;
//case "Parameter Unit": color = red; tagType = TagType.Parameter_Unit; break;
///case "Upper Limit": color = pink; tagType = TestAttributeType.Upper_Limit; break;
///case "Lower Limit": color = pink; tagType = TestAttributeType.Lower_Limit; break;

//case "PatternMode": color = purple; tagType = TestAttributeType.Pattern_Mode_Name; break;
case "Register": color = purple; tagType = TestAttributeType.Pattern_Mode_Register; break;

default: color = grey_default; tagType = TestAttributeType.Default; break;
}

try
{
if
(Selected.ActiveCell.GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat)
!= "")
{
CellIndex fromIndex = Selected.ActiveRange.SelectedCellRange.FromIndex;
CellIndex toIndex = Selected.ActiveRange.SelectedCellRange.ToIndex;
var TagList = Tags.ToList();

for (int c = fromIndex.ColumnIndex; c <= toIndex.ColumnIndex; c++) // Iterates through the
selected column range.
{
int index = TagList.FindIndex(item => item.TagCellIndex == new CellIndex(fromIndex.RowIndex,
c));
if (index < 0)
{
string columnName = DataWorkBook.ActiveWorksheet.Cells[fromIndex.RowIndex,
c].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat); //Gets the
selected column name.
```

```
string above = DataWorkBook.ActiveWorksheet.Cells[fromIndex.RowIndex - 1,
c].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat).Replace(" ", "");
string below = DataWorkBook.ActiveWorksheet.Cells[fromIndex.RowIndex + 1,
c].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat).Replace(" ", "");

var newTag = new DataTag(false) { ColumnName = columnName, ColumnIndex = c, RowIndex =
fromIndex.RowIndex, AttributeType = tagType };

string UnitType = ""; string Unit = "";
if (!string.IsNullOrEmpty(above))
Data.Services.UnitService.MatchUnit(above, out UnitType, out Unit);
if (!string.IsNullOrEmpty(below))
Data.Services.UnitService.MatchUnit(below, out UnitType, out Unit);

newTag.UnitType = UnitType;
newTag.Unit = Unit;
Tags.Add(newTag); //Add tthe tag to the global unprocessed tag collection.

Selected.Cells.SetFill(new PatternFill(PatternType.Solid, color, Colors.Transparent));
DataWorkBook.ActiveWorksheet.Cells[new CellRange(DataBegin, c, DataEnding, c)].SetFill(new
PatternFill(PatternType.Solid, color, Colors.Transparent));

}
}
}
}
catch (Exception ex)
{
System.Windows.MessageBox.Show(ex.Message);
}
}
#endregion

#region Build Test Collection and Export Tables
//Builds the Datatable using the DataTag Collection.
private void BuildTestCollection()
{
var TempDataTable = new DataTable();

for (int i = 0; i <= NumOfTests; i++) //Adds the number of tests/rows.
{
TempDataTable.Rows.Add(TempDataTable.NewRow());
```

```
}

var NewTags = new ObservableCollection<DataTag>(Tags);
//Generates the necessary tags if they're not tagged.
foreach (var e in from TestAttributeType e in tagTypeArray where
NewTags.ToList().FindIndex(item => item.AttributeType == e) < 0 select e)
{
NewTags.Add(new DataTag(true) { ColumnName = e.ToString(), ColumnIndex = 0, AttributeType
= e });
}

NewTags = new ObservableCollection<DataTag>(NewTags.OrderBy(x => x.ColumnIndex));

foreach (TestAttributeType e in tagTypeArray) //Ensures AutoGen tags are in the correct position.
if(e != NewTags[Array.IndexOf(tagTypeArray, e)].AttributeType)
{
int index = NewTags.ToList().FindIndex(item => item.AttributeType == e);
if (index >= 0)
NewTags.Move(index, (int)e);
}

foreach (DataTag tag in NewTags) // Begins writing tagged data columns to the data table.
{
string columnHeader;
var ParamMapTag = new ParameterMapTag() { IsAutoGen = tag.IsAutoGen, AttributeType =
tag.AttributeType };
int DTcolumnIndex = ParamMapTag.ColumnIndex = TempDataTable.Columns.Count;
columnHeader = ParamMapTag.ColumnName = tag.ColumnName;
ParamMapTag.UnitType = tag.UnitType;
ParamMapTag.Unit = tag.Unit;

//Add the columns to the Test Parameters
ParentProject.AddColumn(columnHeader, tag.AttributeType == TestAttributeType.Test_Number ?
typeof(int) : typeof(string), ParamMapTag);
TempDataTable.Columns.Add();

if (tag.IsAutoGen == false)
{
for (int r = 0; r <= NumOfTests; r++) // Iterates through the number of tests/rows to write tagged
data to the data table.
{
bool isMerged = DataWorkBook.ActiveWorksheet.Cells.GetIsMerged(new CellIndex(DataBegin +
```

```
r, tag.ColumnIndex));
if (isMerged == true)
{
CellIndex mergedCellIndex = new CellIndex(DataBegin + r, tag.ColumnIndex);
CellRange mergedCellRange;
bool canGetContainingMergedCellRange =
DataWorkBook.ActiveWorksheet.Cells.TryGetContainingMergedRange(mergedCellIndex, out
mergedCellRange);
TempDataTable.Rows[r][DTcolumnIndex] =
DataWorkBook.ActiveWorksheet.Cells[mergedCellRange.FromIndex.RowIndex,
tag.ColumnIndex].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat);
}
else
{
string valueToStore = DataWorkBook.ActiveWorksheet.Cells[DataBegin + r,
tag.ColumnIndex].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat);
TempDataTable.Rows[r][DTcolumnIndex] = tag.AttributeType == TestAttributeType.Test_Number
? (object)int.Parse(valueToStore) : valueToStore;
}
}
}
else //Is AutoGen.
{
for (int r = 0; r <= NumOfTests; r++) // Iterates through the number of tests/rows to write tagged
data to the data table.
{
TempDataTable.Rows[r][DTcolumnIndex] = ""; //Default value to empty string, cannot leave value
to transfer as null;
}
}

}

var Tests = new ObservableCollection<Test>();
//Build the ITest objects for the Test Collection (uses pre-existing data transfer to DataTable,
change for direct data transfer to ITest(s) later).
int id = 1;
foreach (DataRow r in TempDataTable.Rows)
{
var TestParameters = new ObservableCollection<TestParameter>();
foreach (object o in r.ItemArray)
TestParameters.Add(new TestParameter() { Value = o });
```

```
Tests.Add(new Test(testID: id)
{
TestNumber = int.Parse(r[0].ToString()),
TestName = r[1].ToString(),
Parameters = TestParameters
});
id++; //increment ID.
}

//Order test numbers from least to greatest then add them to DataStorage.
Tests = new ObservableCollection<Test>(Tests.OrderBy(x => x.TestNumber));
Test prevTest = new Test(testID: 0) { TestNumber = 0 }; //Temp.
foreach (Test t in Tests)
{
if (prevTest.TestNumber != t.TestNumber - 1) //If test numeration is non-sequential.
t.IsTestNumBreak = true; //Make current test a numeration break.

ParentProject.Tests.Add(t);
prevTest = t;
}
}
public ICommand ExportTablesCommand { get { return new DelegateCommand(ExportTables); } }
private void ExportTables(object obj)
{
//string cpf = (_viewModelLocator as ViewModelLocator).MWVM.CurrentProjectFile;
if (ParentProject is null) { MessageBox.Show("Project file is Null, Please load a project before
importing."); return;  }

try //temporary try-catch block
{
ParentProject.ResetTable();

this.BuildTestCollection();
ParentProject.BuildSequenceCollection();
ParentProject.AllTestGroupParametersCheck();

//TEMP CODE Set save required on the first PVM in each folder for Test Parameters and Test
Sequences.
PVM testParams =
(PVM)ParentProject.GetProjectFolder(ProjectFileSection.Test_Parameters).FolderItems.FirstOrDe
fault();
```

```
if(testParams != null) { testParams.IsSaveRequired = true; }
PVM dutTestSeq =
(PVM)ParentProject.GetProjectFolder(ProjectFileSection.Test_Sequences).FolderItems.FirstOrDe
fault();
if (dutTestSeq != null) { dutTestSeq.IsSaveRequired = true; }

//(obj as RadWindow).Close(); //Close the window.
Console.WriteLine("Successfully constructed project data!");
}
catch (Exception ex)
{
System.Windows.MessageBox.Show(ex.Message);
}
}
#endregion

#region Build Digital Pattern Model from Tagged Data
public ICommand BuildDigitalPatternModelCommand { get { return new
DelegateCommand(BuildDigitalPatternModel); } }
private void BuildDigitalPatternModel(object obj)
{
//string cpf = (_viewModelLocator as ViewModelLocator).MWVM.CurrentProjectFile;
if (ParentProject is null) { MessageBox.Show("Project file is Null, Please load a project before
importing."); return; }

try
{
var model = new DigitalPatternModel() {  }; ///NEEDS a File Path.

var TempDataTable = new DataTable();

for (int i = 0; i <= NumOfTests; i++)
{
TempDataTable.Rows.Add(TempDataTable.NewRow());
}

foreach (DataTag tag in Tags)
{
//string columnHeader;
//var ParamMapTag = new ParameterMapTag() { IsAutoGen = tag.IsAutoGen, AttributeType =
tag.AttributeType };
int DTcolumnIndex = tag.ColumnIndex = TempDataTable.Columns.Count;
```

```
//columnHeader = ParamMapTag.ColumnName = tag.ColumnName;
//ParamMapTag.UnitType = tag.UnitType;
//ParamMapTag.Unit = tag.Unit;

//Add the columns to the Test Parameters
//PVM.ParentProject.AddColumn(columnHeader, tag.AttributeType ==
TestAttributeType.Test_Number ? typeof(int) : typeof(string), ParamMapTag);
TempDataTable.Columns.Add();

for (int r = 0; r <= NumOfTests; r++) // Iterates through the number of tests/rows to write tagged
data to the data table.
{
bool isMerged = DataWorkBook.ActiveWorksheet.Cells.GetIsMerged(new CellIndex(DataBegin +
r, tag.ColumnIndex));
if (isMerged == true)
{
CellIndex mergedCellIndex = new CellIndex(DataBegin + r, tag.ColumnIndex);
CellRange mergedCellRange;
bool canGetContainingMergedCellRange =
DataWorkBook.ActiveWorksheet.Cells.TryGetContainingMergedRange(mergedCellIndex, out
mergedCellRange);
TempDataTable.Rows[r][DTcolumnIndex] =
DataWorkBook.ActiveWorksheet.Cells[mergedCellRange.FromIndex.RowIndex,
tag.ColumnIndex].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat);
}
else
{
string valueToStore = DataWorkBook.ActiveWorksheet.Cells[DataBegin + r,
tag.ColumnIndex].GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat);
TempDataTable.Rows[r][DTcolumnIndex] = valueToStore;
}
}
}

Tags.Remove(Tags.FirstOrDefault()); //removes the first tag as it is just the name column for the
pattern modes
var modes = new ObservableCollection<Mode>();
foreach (DataRow r in TempDataTable.Rows)
{
var regData = new ObservableCollection<ModeVector>();
//foreach (object o in r.ItemArray)
//    regData.Add(o as string);
```

```
for (int i = 1; i < r.ItemArray.Length; i++) //Start at index one to skip the name column.
{
string inputValue = r.ItemArray[i] as string;
regData.Add(new ModeVector(inputValue));
}

modes.Add(new Mode()
{
Name = Regex.Replace(r[0].ToString(), "[^a-zA-Z0-9_]+", "_"), //Replace all special characters in the mode name for compiler reasons "parenthesis".,
//RegData = regData,
ModeVectors = regData
});
}
model.ModeCollection = modes; //May need Add range rather than complete replace.
model.Registers = new ObservableCollection<string>(Tags.Select(x => x.ColumnName).ToList());
// May need to remove the first tag since its the name

model.Vectorize_All_Modes_For_Pattern(); //TEMP ??
model.CompileAllModes(); //TEMP.

#region Remove the register
//ObservableCollection<string> newRegisters = new ObservableCollection<string>();
////List<string> removeStrings = new List<string>() { "Register", "register", "reg", "regis" }; //This approach uses a list of whitelisted words to replace if they exist in the tagged register string.
//for (int regStr = 0; regStr < model.Registers.Count; regStr++)
//{
//    string newRegisterString = model.Registers[regStr].Replace("Register", string.Empty);
//Spaces ???
//    newRegisters.Insert(regStr, newRegisterString);
//    //for(int i = 0; i < removeStrings.Count; i++)
//    //{
//    //    string newRegisterName = model.Registers[regStr].Replace("Register", string.Empty);
//    //}
//}
//model.Registers = newRegisters;
#endregion

//Need some sort of PVM factory that uses IFile Data as the main input ???
#region PVM and Path Creation
var digiFolder = ParentProject.GetProjectFolder(ProjectFileSection.Digital_Patterns);
string dpuniqueFileName = MainWindowViewModel.GetUniqueFileName(digiFolder, "Digital
```

```
Pattern");
string dpfilePath =
Path.Combine(digiFolder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Digital_
Patterns), $"{dpuniqueFileName}{".rffepat"}");
model.FilePath = dpfilePath;

var dpmPane = new
PatternViewModels.DigitalPatternVM(typeof(UserControls.Patterns.DigitalPattern), dpfilePath,
digiFolder.ParentProject, digiFolder) { IsDocument = true, ParentProject =
digiFolder.ParentProject, IsSaveRequired = false };
dpmPane.CurrentPattern = model;
dpmPane.IsSaveRequired = true;

digiFolder.FolderItems.Add(dpmPane);
digiFolder.IsExpanded = true;
digiFolder.ParentProject.DigitalPatterns.Add(model);

dpmPane.ParentProject.SaveProjectFile(dpmPane.ParentProject.ProjectFilePath);
#endregion PVM and Path Creation

Console.WriteLine("Successfully constructed RFFE digtal pattern!");
}
catch (Exception ex)
{
System.Windows.MessageBox.Show(ex.Message);
}
}
#endregion

#region Build All Tagged Data
public ICommand BuildAllTaggedDataCommand { get { return new
DelegateCommand(BuildAllTaggedData); } }
private void BuildAllTaggedData(object obj)
{
//Add a property for the index of the SHEET that tagged data is located inside the DataTag.

//Need VM strcture to match the case of looping through all the sheets in the excel book imported.
}
#endregion Build All Tagged Data

#region Delete Data Tag
public ICommand DeleteTagCommand { get { return new DelegateCommand(DeleteTag); } }
```

```
//Only works once or so, needs fixing!!! (Note: Works more than once upon deleting Tagged Mode options)
private void DeleteTag(object obj)
{
////Redo to delete from the list of tags.
//if(SelectedTag != null)
//{
//
//}

if (Selected.ActiveCell.GetValue().Value.GetResultValueAsString(CellValueFormat.GeneralFormat) != "")
{
var TagList = Tags.ToList();
CellIndex currentCellIndex = Selected.ActiveCellIndex;
int index = TagList.FindIndex(item => item.TagCellIndex == currentCellIndex);
if (index >= 0)
{
DataTag Rtag = Tags[index];

Tags.Remove(Rtag);

if (Rtag.AttributeType == TestAttributeType.Parameter_Unit)
Selected.Cells.SetFill(new PatternFill(PatternType.Solid, Colors.Transparent, Colors.Transparent));
else
{
Selected.Cells.SetFill(new PatternFill(PatternType.Solid, Colors.Transparent, Colors.Transparent));
DataWorkBook.ActiveWorksheet.Cells[new CellRange(DataBegin, currentCellIndex.ColumnIndex, DataEnding, currentCellIndex.ColumnIndex)].SetFill(new PatternFill(PatternType.Solid, Colors.Transparent, Colors.Transparent));
}
}
}
}
#endregion

#region Reset Tags
public ICommand ResetTagsCommand { get { return new DelegateCommand(ResetTags); } }
private void ResetTags(object obj)
```

```
{
ResetTests();
DataWorkBook.ActiveWorksheet.Cells[DataWorkBook.ActiveWorksheet.UsedCellRange].SetFill(n
ew PatternFill(PatternType.Solid, Colors.Transparent, Colors.Transparent));
}
#endregion

#endregion

#region Methods
private void NumerateRows()
{

}

#endregion

}
}
```

DllFileInfoVM.cs

```
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public interface IDLLInfoView
{

}

public class DllFileInfoVM : PVM
{
private IDLLInfoView _view;
public IDLLInfoView View
```

```
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");
if (value != null)
{
//View.RebindTestLimitsGridView();
//this.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}

private DllFile _dllFileDataObj = new DllFile();
public DllFile DllFileDataObj
{
get { return _dllFileDataObj; }
set
{
_dllFileDataObj = value;
OnPropertyChanged("DllFileDataObj");
if (value != null)
{
SelectedType = value.Types.FirstOrDefault() != null ? value.Types.FirstOrDefault() : null;
}
}
}

private DllType _selectedType;
public DllType SelectedType
{
get { return _selectedType; }
set
{
_selectedType = value;
OnPropertyChanged("SelectedType");
if (value != null)
{
SelectedMethod = value.Methods.FirstOrDefault() != null ? value.Methods.FirstOrDefault() : null;
}
}
```

```
}

private MethodInfo _selectedMethod;
public MethodInfo SelectedMethod
{
get { return _selectedMethod; }
set { _selectedMethod = value; OnPropertyChanged("SelectedMethod");
OnPropertyChanged("SelectedMethodParameters"); }
}

public List<ParameterInfo> SelectedMethodParameters { get { return SelectedMethod != null ?
SelectedMethod.GetParameters().ToList() : new List<ParameterInfo>(); } }

#region Constructor
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;

public override void PaneClose()
{
//this.CommitEditBeforeSave -= View.ForceCommitEdit;
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
}
}

public DllFileInfoVM(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;

this.LoadData();
```

```
this.ChangeOccured += ChangeOccuredSubscriber;
}

public override void SaveData()
{
//CommitEditBeforeSave?.Invoke();

//this.DllFileDataObj.SaveToXml(this.DataFilePath);
//Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public override void LoadData()
{
DllFile dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = new DllFile(this.DataFilePath);
}

if (dataModel != null)
{
this.DllFileDataObj = dataModel;
}

base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{

}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
#endregion
```

}
}

MainWindowViewModel.cs

```
using AppSoftware.LicenceEngine.Common;
using AppSoftware.LicenceEngine.KeyVerification;
using MerlinTest.Common.Types;
using MerlinTest.Tools.Diagnostic.Logging;
using MerlinTest.Tools.TestStudio.Model;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.AppModels;
using MerlinTestStudio_Demo_Telerik.Data.Models.CalModels;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.Data.Models.TestModels;
using MerlinTestStudio_Demo_Telerik.GraphViewModels;
using MerlinTestStudio_Demo_Telerik.UserControls;
using MerlinTestStudio_Demo_Telerik.UserControls.Calibration;
using MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager;
using MerlinTestStudio_Demo_Telerik.UserControls.PaneViews;
using MerlinTestStudio_Demo_Telerik.UserControls.Patterns;
using MerlinTestStudio_Demo_Telerik.UserControls.TestLimits;
using MerlinTestStudio_Demo_Telerik.ViewModels.CalibrationViewModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.DialogViewModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using MT.TestStudio.GUI;
using MT.TestStudio.GUI.User_Controls;
using MT.TestStudio.GUI.ViewModels;
using SFP_UserControlLibrary.Views;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Diagnostics;
using System.Dynamic;
using System.IO;
using System.IO.IsolatedStorage;
using System.Linq;
```

```
using System.Reflection;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Threading;
using System.Xml.Linq;
using System.Xml.Serialization;
using Telerik.Windows;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Docking;
using Unity;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
/// <summary>
/// An interface implementing the Supervising Controller pattern that allows the View to have some controls created dynamically.
/// </summary>
public interface IView
{
RadDocking GetRadDocking();
}

/// <summary>
/// Class that represents the View Model.
/// </summary>
public class MainWindowViewModel : UcViewModelBase
{
#region Private Member Variables
/// <summary>
/// Status from the Model.
/// </summary>
private string status;

/// <summary>
/// Gets / Sets the name of the main window title.
/// </summary>
private string windowTitle;

/// <summary>
/// Gets / Sets the color for the non-telerik controls background.
```

```
/// </summary>
private Brush nonTelerikControlBackGroundColor;

private RecentData _myRecentData = new RecentData();
#endregion Private Member Variables

#region Public Properties

//Should this be apart of preferences ??
private bool _isDevModeActive = false;
public bool IsDevModeActive
{
get { return _isDevModeActive; }
set { _isDevModeActive = value; OnPropertyChanged("IsDevModeActive"); }
}

public static string UserDialogEntry { get; set; }

/// <summary>
/// An instance of the IView interface. This controls access to the VIEW from the VIEWMODEL using the supervising controller pattern.
/// </summary>
public IView View { get; set; }

/// <summary>
/// Status from the Model.
/// </summary>
public string Status
{
get { return status; }
set { status = value; OnPropertyChanged("Status"); }
}

/// <summary>
/// Gets / Sets the color for the non-telerik controls background.
/// </summary>
public Brush NonTelerikControlBackGroundColor
{
get { return nonTelerikControlBackGroundColor; }
set { nonTelerikControlBackGroundColor = value;
OnPropertyChanged("NonTelerikControlBackGroundColor"); }
}
```

```
/// <summary>
/// Gets / Sets the name of the main window title.
/// </summary>
public string WindowTitle
{
get { return "Merlin Test Studio"; } //return string.Format("Merlin Test Studio {0}", windowTitle);
set { windowTitle = value;  OnPropertyChanged("WindowTitle"); }
}

public ObservableCollection<Error> ErrorLog => DataStorage.ErrorLogs;

private ObservableCollection<PVM> _panes = new ObservableCollection<PVM>(); //TEST
public ObservableCollection<PVM> Panes { get { return _panes; } set { _panes = value;
OnPropertyChanged("Panes"); } } // TEST
//public ObservableCollection<PVM> Panes => DataStorage.Panes;
private ObservableCollection<MerlinProject> _projects = new
ObservableCollection<MerlinProject>();
public ObservableCollection<MerlinProject> ProjectsCollection
{
get { return _projects; } set { _projects = value; OnPropertyChanged("ProjectsCollection"); }
}
public ObservableCollection<CalDataModel> CalibrationResultsCollection =>
DataStorage.CalDataResults;



public UserPreferencesData Preferences
{
get { return DataStorage.CurrentUserPreferences; }
set { DataStorage.CurrentUserPreferences = value; OnPropertyChanged("Preferences"); }
}

public IEnumerable<string> TargetSystemOptions { get { return
BackendConstants.TargetSystemOptions; } }

public MerlinSystem MySystem //Selected System
{
get { return DataStorage.CurrentSystem; }
set { DataStorage.CurrentSystem = value; OnPropertyChanged("MySystem"); }
}

public RecentData MyRecentData
```

```
{
get { return _myRecentData; }
set { _myRecentData = value; OnPropertyChanged("MyRecentData"); }
}

/// <summary>
/// Field that is used for diagnostic logging.
/// </summary>
public NetworkTraceSource trace;

#endregion

#region Licence Helpers

private Visibility _Windowvisi = Visibility.Hidden;
public Visibility MainWindowUnlocked
{
get { return _Windowvisi; }
set
{
if (value != _Windowvisi)
{
_Windowvisi = value;
OnPropertyChanged("MainWindowUnlocked");
}
}
}

public static RadWindow LicenceKeyWindow = new RadWindow
{
Width = 800,
Height = 600,
WindowStartupLocation = WindowStartupLocation.CenterScreen,
Content = new UserControls.LicenceManager(),
Header = "Licence Manager"
};

private string _inputLicencekey;
public string InputLicenceKeyText
{
get { return _inputLicencekey; }
set
```

```
{
if (value != _inputLicencekey)
{
_inputLicencekey = value;
OnPropertyChanged("InputLicenceKeyText");
}
}
}

private string _licenceKeyValidationeText;
public string LicenceKeyValidationText
{
get { return _licenceKeyValidationeText; }
set
{
if (value != _licenceKeyValidationeText)
{
_licenceKeyValidationeText = value;
OnPropertyChanged("LicenceKeyValidationText");
}
}
}

private string _licenceKeySeed;
public string LicenceKeySeed
{
get { return _licenceKeySeed; }
set
{
if (value != _licenceKeySeed)
{
_licenceKeySeed = value;
OnPropertyChanged("LicenceKeySeed");
}
}
}

#endregion

#region Constructor

/// <summary>
```

```
/// Constructor for the MainWindow View-Model.
/// </summary>
public MainWindowViewModel()
{
try
{
// Subscribe to the unhandled exception event so that we always handle an unexpected error.
AppDomain.CurrentDomain.UnhandledException += new
UnhandledExceptionEventHandler(CurrentDomain_UnhandledException);

// Don't execute the viewmodel code if we're just looking at the XAML in the designer.
if (DesignerProperties.GetIsInDesignMode(new DependencyObject())) return;

this.trace = new NetworkTraceSource("Merlin Test Studio", "GUI");

#if !DEBUG

// Start Splash Screen
Splash.BeginDisplay();

// Setting the status to show the application is still loading data
Splash.Loading("Loading...");

System.Threading.Thread.Sleep(1200);
#endif

OpenHelpWindow = new DelegateCommand(openHelpWindow);
CheckLicenceKey = new DelegateCommand(LicenceKeyCheck);
NonTelerikControlBackGroundColor = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#252526"));

#region Load User Preferences
//Load user preferences data.
if (File.Exists(BackendConstants.UserPreferencesDataFilePath))
{
Preferences =
UserPreferencesData.LoadFromXml(BackendConstants.UserPreferencesDataFilePath);
}
else
{
Preferences.SaveToXml(BackendConstants.UserPreferencesDataFilePath);
}
```

```
#endregion

#region Load Systems
if (File.Exists(BackendConstants.SystemFilePath))
{
//Should load all systems.
//Use user preferences to know which one is set active.
MySystem = MerlinSystem.ImportXMLNoModifyCheck(BackendConstants.SystemFilePath);
//Each project should be assigned the correct deserialized system on load.
}
else
{
this.MySystem.RF_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "RF110 (1)", InstrumentType = MTSInstrumentType.RF_Instrument });
this.MySystem.RF_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "RF210 (1)", InstrumentType = MTSInstrumentType.RF_Instrument });
this.MySystem.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PE32H (1)", InstrumentType = MTSInstrumentType.PXI });
this.MySystem.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PE32H (2)", InstrumentType = MTSInstrumentType.PXI });
this.MySystem.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "SEDPin32 (1)", InstrumentType = MTSInstrumentType.PXI });
this.MySystem.PXI_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "SEDPin32 (2)", InstrumentType = MTSInstrumentType.PXI });
this.MySystem.Power_Instruments.Add(new MerlinTestStudio_Demo_Telerik.Data.Instrument() {
InstrumentName = "PSM (1)", InstrumentType = MTSInstrumentType.PowerSupply });
MySystem.SaveToXml(BackendConstants.SystemFilePath);
}
#endregion

#region Load Recent Projects
//Load recent user data.
if (File.Exists(BackendConstants.RecentsDataFilePath))
{
MyRecentData = RecentData.LoadFromXml(BackendConstants.RecentsDataFilePath);
}
else
{
MyRecentData.SaveToXml(BackendConstants.RecentsDataFilePath);
}
#endregion
#if !DEBUG
```

```
Splash.Loading("Loading...");

System.Threading.Thread.Sleep(750);

// Setting the status to show the application is still loading data
Splash.Loading("Completed");

System.Threading.Thread.Sleep(250);

// As we're loaded remove the spash screen and re-instate the MainWindow
Splash.EndDisplay();
#endif
}
catch (Exception ex)
{
this.trace.TraceError("Start-Up Failure : " + ex.Message, "Merlin Test Studio");
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
#if !DEBUG
// An error occured so take down the splash screen. Make the main window visible, log an error and request the main window be closed.
Splash.EndDisplay();
#endif
}
}

#endregion Constructor

#region Commands

//Currently only works on tool panes.
#region Bring Pane Into View
public ICommand BringPaneIntoViewCommand { get { return new RelayCommand(BringPaneIntoView); } }
private void BringPaneIntoView()
{
try
{
//Needs to be able to bring documents into view as well via Error Double Click
```

```
foreach (PaneViewModel pane in this.Panes.OfType<PaneViewModel>()) //Can Only loop through
side tool panes.
{
if (pane.ContentType == typeof(ErrorList))
{
pane.IsHidden = false;
pane.IsActive = true;
}
}
}
catch(Exception ex)
{
Console.WriteLine(ex.Message);
}
}
#endregion

#region Open Preferences
public ICommand OpenPreferencesCommand { get { return new
RelayCommand(OpenPreferences); } }
private void OpenPreferences()
{
//string header = "Preferences";
//PVM pane = null;
//if (CheckPaneHeaderExists(header, out pane) != true)
//{
//    var prefPanel = new PaneViewModel(typeof(RF110_SFP_View)) { Header = header,
InitialPosition = DockState.DockedLeft, IsDocument = true };
//    CreateNewDocument(prefPanel);
//}
//else { pane.IsActive = true; } //Select Pane.

//OR

RadWindow prefPanel = new RadWindow()
{
Header = "Preferences",
Height = 800,
Width = 1000,
WindowStartupLocation = WindowStartupLocation.CenterScreen,
Content = new UserPreferences(this.Preferences)
};
```

```
prefPanel.Show();
//Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Normal, new Action(() => {
prefPanel.Show(); }));
}
#endregion

#region Change Theme
/// <summary>
/// Saves to .aiq
/// </summary>
public ICommand ChangeThemeCommand { get { return new
DelegateCommand(ChangeTheme); } }

private void ChangeTheme(object obj)
{
try
{
if (obj != null)
{
string themeName = obj as string;

switch (themeName) //Dose not work currently, try placing in the back end of ma in window.
{
case "Dark":
VisualStudio2013Palette.LoadPreset(VisualStudio2013Palette.ColorVariation.Dark);
break;
case "Light":
VisualStudio2013Palette.LoadPreset(VisualStudio2013Palette.ColorVariation.Light);
break;
case "Blue":
VisualStudio2013Palette.LoadPreset(VisualStudio2013Palette.ColorVariation.Blue);
break;
}
}
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
}
#endregion
```

```
#region Licensing Commands

public ICommand OpenHelpWindow { get; set; }
private void openHelpWindow(object obj)
{
LicenceKeyWindow.Show();
}

public ICommand CheckLicenceKey { get; set; }
private void LicenceKeyCheck(object obj)
{
var result = PkvLicenceKeyResult.KeyInvalid;

string userEnteredLicenceKeyStr = InputLicenceKeyText;

var pkvKeyCheck = new PkvKeyCheck();

// Here we recreate a subset of the full original KeyByteSet array that
// was used to create the licence key. Note the argument to keyByteNo in
// each matches that in the full KeyByteSet array.
var keyBytes = new[] {

new KeyByteSet(5, 165, 15, 132),
new KeyByteSet(6, 128, 175, 213)
};
string extra = userEnteredLicenceKeyStr.Substring(8, 1).ToString();
string hex = (userEnteredLicenceKeyStr.Substring(0, 7) + extra).ToString();
int seed = int.Parse(hex, System.Globalization.NumberStyles.HexNumber);
LicenceKeySeed = seed.ToString();

result = pkvKeyCheck.CheckKey(userEnteredLicenceKeyStr, keyBytes, 8, null);

if (result != PkvLicenceKeyResult.KeyGood)
{
LicenceKeyValidationText = ("Result is: " + result.ToString() + ". Please try again.");
}
if (result == PkvLicenceKeyResult.KeyGood)
{
LicenceKeyValidationText = ("Result is: " + result.ToString());
MainWindowUnlocked = Visibility.Visible;
OnPropertyChanged("MainWindowUnlocked");
```

```
}

}

#endregion

//Experimental.
#region Add/Remove/Duplicate Test Commands
//SHOULD THIS BE AN INSERT COMMAND?
public ICommand AddTestToActiveProjectCommand { get { return new
RelayCommand(AddTestToActiveProject); } }
//Save all projects and their documents.
private void AddTestToActiveProject()
{
Console.WriteLine("Adding new test...");

//Example below...
//this.GetActiveParentProject().AddNewTest();

}

public ICommand RemoveTestFromActiveProjectCommand { get { return new
RelayCommand(RemoveTestFromActiveProject); } }
//Save all projects and their documents.
private void RemoveTestFromActiveProject()
{
Console.WriteLine("Removing test...");

//Example below...
//this.GetActiveParentProject().AddNewTest();

}

public ICommand DuplicateTestToActiveProjectCommand { get { return new
RelayCommand(DuplicateTestToActiveProject); } }
//Save all projects and their documents.
private void DuplicateTestToActiveProject()
{
Console.WriteLine("Duplicating test...");

//Example below...
//this.GetActiveParentProject().AddNewTest();
```

```
}
#endregion Add/Remove/Duplicate Test Commands

#region Save All Documents
public ICommand SaveAllProjectDocumentsCommand { get { return new
RelayCommand(SaveAllProjectDocuments); } }
//Save all projects and their documents.
private void SaveAllProjectDocuments()
{
foreach (MerlinProject mp in this.ProjectsCollection)
{
foreach (PVM pvm in mp.GetAllProjectDocumentPanes())
{
pvm.SaveData();
}

mp.SaveProjectFile(mp.ProjectFilePath);
}

MySystem.SaveToXml(BackendConstants.SystemFilePath);

}
#endregion

#region Current Document Save
public ICommand SaveCurrentDocumentCommand { get { return new
RelayCommand(SaveCurrentDocument); } }
private void SaveCurrentDocument()
{
var pvm = GetActivePane();
if (pvm != null && pvm.IsDocument)
{
pvm.SaveData();
//if(pvm.DataObject != null)
//{
//    pvm.SaveData();
//    pvm.IsSaveRequired = false;
//}
//pvm.ParentProject.SaveProjectFile(pvm.ParentProject.ProjectFilePath); //TEMP
}
}
```

```
#endregion

#region Document Undo
public ICommand DocumentUndoCommand { get { return new RelayCommand(DocumentUndo); }
}
private void DocumentUndo()
{
PVM cPvm = GetActivePane();
if (cPvm.IsDocument)
{
cPvm.Undo();
}
}
#endregion

#region Documnet Redo
public ICommand DocumentRedoCommand { get { return new RelayCommand(DocumentRedo); }
}
private void DocumentRedo()
{
PVM cPvm = GetActivePane();
if (cPvm.IsDocument)
{
cPvm.Redo();
}
}
#endregion

#region Open Data Formatting Tool
/// <summary>
/// Opens a dialog to load Conditions/Limits data into the DataTables.
/// </summary>
public ICommand FormatProjectTestDataCommand { get { return new
DelegateCommand(FormatProjectTestData); } }
private void FormatProjectTestData(object obj)
{
MerlinProject mp = (obj as MerlinProject);
if (mp is null) { Console.WriteLine("The provided project is null, unable to open Data Formatting
tool"); return; }

string header = string.Format("{0} - Data Formatting Tool", mp.ProjectName);
PVM pane = null;
```

```
if (CheckPaneHeaderExists(header, out pane) != true)
{
var resultsViewPane = new
DataFormattingToolViewModel(typeof(UserControls.Tools.DataFormattingTool), mp) { Header =
header, InitialPosition = DockState.DockedLeft, IsDocument = true };
CreateNewDocument(resultsViewPane);
}
else { pane.IsActive = true; } //Select Pane.

//RadWindow DFT = new RadWindow()
//{
//    Header = "Data Formatting Tool",
//    Height = 800,
//    Width = 1000,
//    WindowStartupLocation = WindowStartupLocation.CenterScreen,
//    Content = new UserControls.Tools.DataFormattingTool()
//};
//DFT.Show();
}
#endregion

#region RF110 SFP
public ICommand OpenRF110_SFP_Command { get { return new
RelayCommand(OpenRF110_SFP); } }
private void OpenRF110_SFP()
{
string header = "RF110 SFP";
PVM pane = null;
if (CheckPaneHeaderExists(header, out pane) != true)
{
var rf110sfp = new PaneViewModel(typeof(RF110_SFP_View)) { Header = header, InitialPosition
= DockState.DockedLeft, IsDocument = true };
CreateNewDocument(rf110sfp);
}
else { pane.IsActive = true; } //Select Pane.

///RadWindow SFP = new RadWindow()
///{
///    Header = "RF110 SFP",
///    Height = 800,
///    Width = 1000,
///    WindowStartupLocation = WindowStartupLocation.CenterScreen,
```

```
///    Content = new RF110_SFP_View()
///};
///Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Normal, new Action(() => {
SFP.Show(); }));
}
#endregion
#region RF210 SFP
public ICommand OpenRF210_SFP_Command { get { return new
RelayCommand(OpenRF210_SFP); } }
private void OpenRF210_SFP()
{
string header = "RF210 SFP";
PVM pane = null;
if (CheckPaneHeaderExists(header, out pane) != true)
{
var rf210sfp = new PaneViewModel(typeof(RF210_SFP_View)) { Header = header, InitialPosition
= DockState.DockedLeft, IsDocument = true };
CreateNewDocument(rf210sfp);
}
else { pane.IsActive = true; } //Select Pane.

///RadWindow SFP = new RadWindow()
///{
///    Header = "RF210 SFP",
///    Height = 800,
///    Width = 1000,
///    WindowStartupLocation = WindowStartupLocation.CenterScreen,
///    Content = new RF210_SFP_View()
///};
///Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Normal, new Action(() => {
SFP.Show(); }));
}
#endregion

#region Compile Sequence
public ICommand CompileCommand { get { return new RelayCommand(Compile); } }

private void Compile()
{
//Microsoft.Win32.SaveFileDialog saveFileDlg = new Microsoft.Win32.SaveFileDialog()
//{ Title = "Save Sequence", FileName = "Sequence", Filter = "XML (*.xml)|*.xml" };
//Nullable<bool> result = saveFileDlg.ShowDialog();
```

```
//
//if (result == true)
//{
//    ///Need the selected project from the project tree
//    //MerlinProject.CompileTestSequence(saveFileDlg.FileName, project);
//}
//else { return; }
Console.WriteLine("Compile command is disabled from this menu location, please try compiling from DUT Test.");
}
#endregion

#region Open Merlin Test Project File

/// <summary>
/// Opens a dialog to load a .MTPROJ solution file.
/// </summary>
public ICommand OpenSolutionFileCommand
{
get
{
return new RelayCommand(OpenSolutionFileCommandExecute, CanOpenSolutionFileCommandExecute);
}
}

/// <summary>
/// Opens a dialog to load a .MTPROJ solution file.
/// </summary>
private void OpenSolutionFileCommandExecute()
{
try
{
// Configure open file dialog box
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();
dlg.DefaultExt = BackendConstants.MerlinProjectExtension; // Default file extension
dlg.Filter = "MTPROJ / Legacy MTP Files(.mtproj; .mtp;)|*.mtproj; *.mtp;";
//dlg.Filter = "MTPROJ File(*.mtproj)|*.mtproj;|Legacy MTP File(*.mtp)|*.mtp;"; // Filter files by extension //MTSLN File (.mtsln)|*.mtsln;|
dlg.InitialDirectory = BackendConstants.InitialProjectDirectory;

// Show open file dialog box
```

```
Nullable<bool> result = dlg.ShowDialog();

// Process open file dialog box results
if (result == true)
{
// The code below is just temporary until the open project code has been written.
//string directory =
Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData);
///LoadProjectTree(dlg.FileName);

switch (Path.GetExtension(dlg.FileName))
{
// case BackendConstants.MerlinSolutionExtension:
//     LoadMerlinSolution(dlg.FileName);
//     break;
case BackendConstants.MerlinProjectExtension:
LoadMerlinProject(dlg.FileName);
break;
case ".mtp":
LoadLegacyProjectMTP(dlg.FileName);
break;
}

trace.TraceInformation("File " + dlg.FileName + " succesfully loaded.");

Status = "File " + dlg.FileName + " succesfully loaded.";

//this.WindowTitle = " - " + this.CurrentProductName;
}
}
catch (Exception ex)
{
trace.TraceError("An error occured : " + ex.Message);
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}
}



/// <summary>
/// Can the command execute.
/// </summary>
/// <returns>True (always)</returns>
```

```
private bool CanOpenSolutionFileCommandExecute()
{
return true;
}

#endregion  Open Merlin Test Project File

#region Unload Merlin Project
public ICommand UnloadMerlinProjectCommand { get { return new
DelegateCommand(UnloadMerlinProject); } }
private void UnloadMerlinProject(object obj)
{
try
{
if(obj != null)
{
MerlinProject projectToRemove = obj as MerlinProject;

bool projectClosed = CloseProjects(new List<MerlinProject>() { projectToRemove });
if (projectClosed)
{
this.Status = "Projects closed successfully.";
}

}
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
}
}
#endregion

#region Add Existing Merlin Project
public ICommand AddExistingMerlinProjectCommand { get { return new
DelegateCommand(AddExistingMerlinProject); } }
private void AddExistingMerlinProject(object obj)
{
// Create OpenFileDialog
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog()
{
DefaultExt = ".mtproj",
```

```
Filter = "MTPROJ File (.mtproj)|*.mtproj",
Title = "Add Existing Merlin Project"
};
Nullable<bool> result = dlg.ShowDialog();

if (result == true)
{
LoadMerlinProject(dlg.FileName);
}
}
#endregion

//Opens Window for new project.
#region New Merlin Test Project File

/// <summary>
/// Opens a dialog to create a new .MTPROJ solution file.
/// </summary>
public ICommand NewProjectFileCommand
{
get
{
return new RelayCommand(NewProjectFileCommandExecute,
CanNewProjectFileCommandExecute);
}
}

/// <summary>
/// Opens a dialog to create a new .MTPROJ solution file.
/// </summary>
private void NewProjectFileCommandExecute()
{
try
{
var view = new WindowStyled(this)
{
Content = new NewProjectDiaglog(),
Title = "New Project",
WindowState = WindowState.Normal,
SizeToContent = SizeToContent.Manual,
Width = 945,
Height = 500,
```

```
ResizeMode = ResizeMode.NoResize,
WindowStartupLocation = WindowStartupLocation.CenterScreen
};

//view.DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<NewProjectDiaglogViewModel
>(); //Causes bugs.
view.DataContext = new NewProjectDiaglogViewModel(this, view);

// Display Window.
bool? dialogResult = view.ShowDialog();

// Determine how the Diaglog was closed.
switch (dialogResult)
{
case true:
{
// User clciked OK dialog box

break;
}
case false:
{
// User canceled dialog box
// Do nothing.
break;
}
default:
{
// Indeterminate
break;
}
}
}
catch (Exception ex)
{
trace.TraceError("An error occured : " + ex.Message);
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}
}

// <summary>
```

```
// Can the command execute.
// </summary>
// <returns>True (always)</returns>
private bool CanNewProjectFileCommandExecute()
{
return true;
}

#endregion New Merlin Test Project File

#region Open Recent Merlin Project
public ICommand OpenRecentMerlinProjectCommand { get { return new
DelegateCommand(OpenRecentMerlinProject); } }
private void OpenRecentMerlinProject(object obj)
{
if (obj != null)
{
string filePathParam = obj as string;
LoadMerlinProject(filePathParam);
}
}
#endregion

//Currently used for viewing the project properties.
#region View Properties
public ICommand ViewPropertiesCommand { get { return new
DelegateCommand(ViewProperties); } }
private void ViewProperties(object obj)
{
try
{
if (obj != null)
{
MerlinProject projectPropertiesToView = obj as MerlinProject;

string header = string.Format("{0} - Project Configuration", projectPropertiesToView.ProjectName);
PVM pane = null;
if (CheckPaneHeaderExists(header, out pane) != true)
{
var projectConfigPane = new PaneViewModel(typeof(ProjectConfiguration),
projectPropertiesToView) { Header = header, InitialPosition = DockState.DockedLeft, IsDocument
= true };
```

```
CreateNewDocument(projectConfigPane);
}
else { pane.IsActive = true; } //Select Pane.
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
#endregion

#region Add New File
public ICommand AddNewFileCommand { get { return new DelegateCommand(AddNewFile); } }
private void AddNewFile(object obj) //NEEDS check existing and add overwrite protection
{
if(obj is ProjectFolder)
{
ProjectFolder folder = obj as ProjectFolder;

var vm = new AddNewFileDialogVM(folder);
RadWindow AddNewItemWindow = new RadWindow()
{
Width = 800,
Height = 600,
WindowStartupLocation = System.Windows.WindowStartupLocation.CenterScreen,
Content = new UserControls.Dialogs.AddNewFileDialog(vm),
Header = "Add New..."
};
AddNewItemWindow.ShowDialog();

PVM newPane = null;
if (string.IsNullOrEmpty(vm.DataFilePathProduced))
{
return; //Stop code if no item is produced.
}

switch (folder.Section)
{
case ProjectFileSection.Product_Configurations:
switch (vm.SelectedItem.Extension) //Continue addition process based on selected file extension.
{
case BackendConstants.MerlinProduct_Config_Extension:
newPane = new ProductConfigViewModel(typeof(ProductConfigurationView),
vm.DataFilePathProduced, folder.ParentProject, folder) { IsDocument = true, IsHidden = true, Icon
```

```
= "Settingsd.ico" };
break;
}
break;
case ProjectFileSection.Test_Limits:
///Error message displaying that no base data exists on import....
//if (folder.ParentProject.TestDataDefinitions.Count == 0) //Check if any base limit data might
exist...
//{
//    MessageBox.Show("No base data to collect test information for new file use, Please import a
Test limits file...");
//    return; //Stop code if no base data exists.
//}

switch (vm.SelectedItem.Extension) //Continue addition process based on selected file extension.
{
case BackendConstants.Test_Limits_Extension:
newPane = new TestLimitsVM(typeof(TestLimits), vm.DataFilePathProduced,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject  };
break;
case BackendConstants.Gold_Limits_Extension:
newPane = new GoldLimitsVM(typeof(GoldLimits), vm.DataFilePathProduced,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject };
break;
case BackendConstants.Offset_Limits_Extension:
newPane = new OffsetLimitsVM(typeof(OffsetLimits), vm.DataFilePathProduced,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject };
break;
case BackendConstants.Traceloss_Limits_Extension:
newPane = new TracelossLimitsVM(typeof(TracelossLimits), vm.DataFilePathProduced,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject };
break;
case BackendConstants.Test_Fixture_Limits_Extension:
newPane = new TestFixturesVM(typeof(TestFixtures), vm.DataFilePathProduced,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject };
break;
}

break;
case ProjectFileSection.Test_Parameters:
//SOON
break;
```

```
case ProjectFileSection.Test_Sequences:
//SOON
break;
case ProjectFileSection.Connection_Manager:
break;
case ProjectFileSection.Connection_Diagrams:
switch (vm.SelectedItem.Extension) //Continue addition process based on selected file extension.
{
case ".xml":
newPane = new DiagramViewModel(typeof(RF_Cable_Map), vm.DataFilePathProduced,
folder.ParentProject, folder) { IsDocument = true, IsHidden = true };
break;
}
break;
case ProjectFileSection.Calibration_Definitions:
switch (vm.SelectedItem.Extension) //Continue addition process based on selected file extension.
{
case BackendConstants.MTS_Cal_Config_Extension:
newPane = new CalibrationConfiguratorVM(typeof(CalibrationConfiguratorView),
vm.DataFilePathProduced, folder.ParentProject, folder) { IsDocument = true, ParentProject =
folder.ParentProject };
break;
}
break;
case ProjectFileSection.Waveforms:
//CANNOT ADD NEW WAVEFORMS.
break;
case ProjectFileSection.Digital_Patterns: // Adding files under this section is currently handled in
the AddNewFileDialogVM.
switch (vm.SelectedItem.Extension) //Continue addition process based on selected file extension.
{
case ".genericrffepattern":
newPane = new GenericRffePatternVM(typeof(GenericRffePattern), vm.DataFilePathProduced,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject,
IsSaveRequired = false };
break;
case ".rffepat":
newPane = new DigitalPatternVM(typeof(DigitalPattern), vm.DataFilePathProduced,
folder.ParentProject, folder) {  IsDocument = true, ParentProject = folder.ParentProject,
IsSaveRequired = false };
break;
case ".digilevels":
```

```
newPane = new DigitalLevelViewModel(typeof(DigitalLevel), vm.DataFilePathProduced,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject,
IsSaveRequired = false };
break;
case ".digitiming":
newPane = new DigitalTimingVM(typeof(DigitalTiming), vm.DataFilePathProduced,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject,
IsSaveRequired = false };
break;
}
break;
}

if (newPane != null)
{
newPane.IsSaveRequired = true;
folder.FolderItems.Add(newPane);
folder.IsExpanded = true;
}

folder.ParentProject.SaveProjectFile(folder.ParentProject.ProjectFilePath);
}
}

#endregion

#region Save as .aiq
/// <summary>
/// Saves to .aiq
/// </summary>
public ICommand SaveAsAIQCommand { get { return new DelegateCommand(SaveAsAIQ); } }

private void SaveAsAIQ(object obj)
{
try
{
var wfMVM = (obj as PaneViewModel).ViewDataContext as
WaveformConverterControls.MainWindowViewModel;
this.Status = wfMVM.SaveAsAIQMethod();
//The line below saves to the project waveform file and gets rid of the IsSaveRequired icon.
//(obj as PVM).SaveData();
}
```

```
catch (Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
}
#endregion
#region Save as .mtwf2
/// <summary>
/// Saves to .aiq
/// </summary>
public ICommand SaveAsMTWFCommand { get { return new DelegateCommand(SaveAsMTWF);
} }

private void SaveAsMTWF(object obj)
{
try
{
var wfMVM = (obj as PaneViewModel).ViewDataContext as
WaveformConverterControls.MainWindowViewModel;
this.Status = wfMVM.SaveAsMTWFMethod();
//The line below saves to the project waveform file and gets rid of the IsSaveRequired icon.
//(obj as PVM).SaveData();
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
}
#endregion

#region Add Existing File
public ICommand AddExistingFileCommand { get { return new
DelegateCommand(AddExistingFile); } }
private void AddExistingFile(object obj)
{
try
{
if (obj is ProjectFolder)
{
ProjectFolder folder = obj as ProjectFolder;
```

```
Microsoft.Win32.OpenFileDialog open_file_dlg = new Microsoft.Win32.OpenFileDialog()
{
//No width or height properties???
Multiselect = true
};

//Set the dialog settings and filters.
switch (folder.Section)
{
case ProjectFileSection.Product_Configurations:
open_file_dlg.DefaultExt = BackendConstants.MerlinProduct_Config_Extension;
open_file_dlg.Filter = $"Product Config
({BackendConstants.MerlinProduct_Config_Extension};)|*{BackendConstants.MerlinProduct_Confi
g_Extension}; ";
open_file_dlg.Title = "Add Existing Product Configuration";
break;
case ProjectFileSection.Test_Limits:
open_file_dlg.DefaultExt = ".limits";
//open_file_dlg.Filter = "XML / CSV Files(.xml; .csv;)|*.xml; *.csv| All Limit Files(.limits; .golds;
.offsets; .traceloss; .fixtures;)|*.limits; *.golds; *.offsets; *.traceloss; *.fixtures";
open_file_dlg.Filter = "All Limit Files(.limits; .golds; .offsets; .traceloss; .fixtures;)|*.limits; *.golds;
*.offsets; *.traceloss; *.fixtures";
open_file_dlg.Title = "Add Existing Test Limit Files";
break;
case ProjectFileSection.Test_Parameters:
break;
case ProjectFileSection.Test_Sequences:
case ProjectFileSection.Connection_Manager:
//Do nothing... Control Folders.
break;
case Data.ProjectFileSection.Connection_Diagrams:
open_file_dlg.DefaultExt = $".xml";
open_file_dlg.Filter = $"Connection Diagram (.xml;)|*.xml; ";
open_file_dlg.Title = "Add Connection Diagrams";
break;
case ProjectFileSection.Calibration_Definitions:
open_file_dlg.DefaultExt = BackendConstants.MTS_Cal_Config_Extension;
open_file_dlg.Filter = $"CALDEF
({BackendConstants.MTS_Cal_Config_Extension};)|*{BackendConstants.MTS_Cal_Config_Exten
sion}; ";
//open_file_dlg.Filter = "JSON / XML / CSV Files(.json; .xml; .csv;)|*.json; *.xml; *.csv";
```

```
open_file_dlg.Title = "Add Existing Cal Def";
break;
case ProjectFileSection.Calibration_Limits:
open_file_dlg.DefaultExt = $"{BackendConstants.Cal_Limit_Extension}";
open_file_dlg.Filter = $"Cal Limit
({BackendConstants.Cal_Limit_Extension};)|*{BackendConstants.Cal_Limit_Extension}; ";
open_file_dlg.Title = "Add Existing Calibration Limits";
break;
case ProjectFileSection.Waveforms:
open_file_dlg.DefaultExt = ".mtwf2";
open_file_dlg.Filter = "MTWF Files(.mtwf2)|*.mtwf2";
open_file_dlg.Title = "Add Existing MTWF Waveforms";
//Change the allowed waveforms file format extension in the project if necessary.
if(folder.ParentProject.UserTargetSystem != "APS-500")
{
open_file_dlg.DefaultExt = ".aiq"; open_file_dlg.Filter = "AIQ Files(.aiq;)|*.aiq;"; open_file_dlg.Title
= "Add Existing AIQ Waveforms";
}
break;
case ProjectFileSection.Digital_Patterns:
open_file_dlg.DefaultExt = ".rffepat";
open_file_dlg.Filter = "Digital Pattern Files(.genericrffepattern; .rffepat; .digitiming;
.digilevels;)|*.genericrffepattern; *.rffepat; *.digitiming; *.digilevels";
open_file_dlg.Title = "Add Existing Digital Pattern Files";
break;
case ProjectFileSection.Dll:
open_file_dlg.DefaultExt = ".dll";
open_file_dlg.Filter = "DLL (.dll;)|*.dll;";
open_file_dlg.Title = "Add Reference";
break;
}

if (string.IsNullOrEmpty(open_file_dlg.Filter)) { return; } //No options, Don't show File Dialog.

//Show dialog and get file path(s).
if (open_file_dlg.ShowDialog() == true)
{
List<string> dialogFileNamesList = new List<string>(open_file_dlg.FileNames);
List<string> outputFilePaths = new List<string>();

//Process User Selected File Paths to Output Paths for overwrite check.
switch (folder.Section)
```

```
{
case ProjectFileSection.Product_Configurations:
foreach (string filename in open_file_dlg.FileNames)
{
string fileNameNoExt = Path.GetFileNameWithoutExtension(open_file_dlg.FileName);
string copyOutputPath =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Product_Co
nfigurations), $"{fileNameNoExt}{BackendConstants.MerlinProduct_Config_Extension}");
outputFilePaths.Add(copyOutputPath);
}
break;
case ProjectFileSection.Test_Limits:
foreach (string filename in open_file_dlg.FileNames)
{
string fileNameNoExtension = Path.GetFileNameWithoutExtension(filename);
string fileExt = Path.GetExtension(filename);

switch (fileExt)
{
case ".csv": //For legacy imports.
case ".xml":
string copyOutputPath_Limits =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Test_Limits)
, $"{fileNameNoExtension}.limits");
outputFilePaths.Add(copyOutputPath_Limits);
string copyOutputPath_Golds =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Test_Limits)
, $"{fileNameNoExtension}.golds");
outputFilePaths.Add(copyOutputPath_Golds);
string copyOutputPath_Offsets =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Test_Limits)
, $"{fileNameNoExtension}.offsets");
outputFilePaths.Add(copyOutputPath_Offsets);
break;
case ".limits":
case ".golds":
case ".offsets":
case ".traceloss":
case ".fixtures":
string limitFilePath =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Test_Limits)
, Path.GetFileName(filename));
```

```
outputFilePaths.Add(limitFilePath);
break;
}
}
break;
case ProjectFileSection.Test_Parameters:
break;
case ProjectFileSection.Test_Sequences:
case ProjectFileSection.Connection_Manager:
//Do nothing... Control Folders.
break;
case Data.ProjectFileSection.Connection_Diagrams:
foreach (string filename in open_file_dlg.FileNames)
{
string fileNameNoExt = Path.GetFileNameWithoutExtension(open_file_dlg.FileName);
string copyOutputPath =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Connection
_Diagrams), $"{fileNameNoExt}.xml");
outputFilePaths.Add(copyOutputPath);
}
break;
case ProjectFileSection.Calibration_Definitions:
foreach (string filename in open_file_dlg.FileNames)
{
string fileNameNoExt = Path.GetFileNameWithoutExtension(open_file_dlg.FileName);
string copyOutputPath =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Project_Dat
a), "Cal Data", $"{fileNameNoExt}{BackendConstants.MTS_Cal_Config_Extension}");
outputFilePaths.Add(copyOutputPath);
}
break;
case ProjectFileSection.Calibration_Limits:
foreach (string filename in open_file_dlg.FileNames)
{
string fileNameNoExt = Path.GetFileNameWithoutExtension(open_file_dlg.FileName);
string copyOutputPath =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Calibration_
Limits), $"{fileNameNoExt}{BackendConstants.Cal_Limit_Extension}");
outputFilePaths.Add(copyOutputPath);
}
break;
case ProjectFileSection.Waveforms:
```

```
foreach (string filename in open_file_dlg.FileNames)
{
//Default output path for the waveform project directory.
string copyOutputPath =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Waveforms)
, Path.GetFileName(filename));

string eDir = MerlinProject.ExtractWaveformSubDirectoryFromFilePath(filename);
if (!string.IsNullOrEmpty(eDir)) //if the technology directory exists in merlin test studio.
{
string techDirectory =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Waveforms)
, eDir);
//Create directory if it does not already exist.
if (!Directory.Exists(techDirectory)) { Directory.CreateDirectory(techDirectory); }

//Change the default output path to the correct waveform tech sub directory.
copyOutputPath = Path.Combine(techDirectory, Path.GetFileName(filename));
}
else { } //Else it spits out the waveform in a unspecified technology folder.

//// If the file already exists in the project's waveform folder AND in the project tree then skip
//if (File.Exists(copyOutputPath) && folder.ParentProject.Waveforms.Any(wf => wf.FilePath ==
copyOutputPath))
//{
//    MessageBox.Show("Duplicate waveforms cannot be added to the project", "Merlin Test
Studio", MessageBoxButton.OK, MessageBoxImage.Warning);
//    continue;
//}

outputFilePaths.Add(copyOutputPath);
}
break;
case ProjectFileSection.Digital_Patterns:
foreach (string filename in open_file_dlg.FileNames)
{
string copyOutputPath =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Digital_Patt
erns), Path.GetFileName(filename));
string justOutputFileName = Path.GetFileName(copyOutputPath);
outputFilePaths.Add(copyOutputPath);
}
```

```
break;
case ProjectFileSection.Dll:
foreach (string filename in open_file_dlg.FileNames)
{
string fileNameNoExt = Path.GetFileNameWithoutExtension(open_file_dlg.FileName);
string copyOutputPath =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Dll),
$"{fileNameNoExt}.dll");
outputFilePaths.Add(copyOutputPath);
}
break;
}

//iterate through the dialog file names and match with copy output file names.
for (int i = 0; i < dialogFileNamesList.Count(); i++)
{
string inputPath = dialogFileNamesList[i];
string outputPath = outputFilePaths[i];
string outFileName = Path.GetFileName(outputPath);

#region New Overwrite Check
Action copyAction = () =>
{
File.Copy(inputPath, outputPath, true);
};
bool wasOverwritePrompted = OverwriteCheck(folder, inputPath, outputPath, copyAction);
if(wasOverwritePrompted == true)
{
continue;
}
#endregion

IFileData dataObj = null;
PVM newPaneVM = null;

//Create the panes user selected file paths after overwrite check.
switch (folder.Section)
{
case ProjectFileSection.Product_Configurations:
switch (Path.GetExtension(inputPath))
{
case BackendConstants.MerlinProduct_Config_Extension:
```

```
newPaneVM = new ProductConfigViewModel(typeof(ProductConfigurationView), outputPath,
folder.ParentProject, folder) { IsDocument = true, IsHidden = true, Icon = "Settingsd.ico" };
break;
}
break;
case ProjectFileSection.Test_Limits:
//.xml and .csv does will not work to extract three PVMs with one file.
#region Test Limits PVM
switch (Path.GetExtension(inputPath))
{
case ".limits":
newPaneVM = new TestLimitsVM(typeof(TestLimits), outputPath, folder.ParentProject, folder) {
IsDocument = true, ParentProject = folder.ParentProject, IsHidden = true };
break;
case ".golds":
newPaneVM = new GoldLimitsVM(typeof(GoldLimits), outputPath, folder.ParentProject, folder) {
IsDocument = true, ParentProject = folder.ParentProject, IsHidden = true };
break;
case ".offsets":
newPaneVM = new OffsetLimitsVM(typeof(OffsetLimits), outputPath, folder.ParentProject, folder) {
IsDocument = true, ParentProject = folder.ParentProject, IsHidden = true };
break;
case ".traceloss":
newPaneVM = new TracelossLimitsVM(typeof(TracelossLimits), outputPath, folder.ParentProject,
folder) { IsDocument = true, ParentProject = folder.ParentProject, IsHidden = true };
break;
case ".fixtures":
newPaneVM = new TestFixturesVM(typeof(TestFixtures), outputPath, folder.ParentProject, folder)
{ IsDocument = true, ParentProject = folder.ParentProject, IsHidden = true };
break;
}
#endregion Test Limits PVM
break;
case ProjectFileSection.Test_Parameters:
break;
case ProjectFileSection.Test_Sequences:
case ProjectFileSection.Connection_Manager:
//Do nothing... Control Folders.
break;
case Data.ProjectFileSection.Connection_Diagrams:
switch (Path.GetExtension(inputPath))
{
```

```
case ".xml":
newPaneVM = new DiagramViewModel(typeof(RF_Cable_Map), outputPath, folder.ParentProject,
folder) { IsDocument = true, IsHidden = true };
break;
}
break;
case ProjectFileSection.Calibration_Definitions:
switch (Path.GetExtension(inputPath))
{
case BackendConstants.MTS_Cal_Config_Extension:
newPaneVM = new CalibrationConfiguratorVM(typeof(CalibrationConfiguratorView), outputPath,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject, IsHidden =
true };
break;
}
break;
case ProjectFileSection.Calibration_Limits:
switch (Path.GetExtension(inputPath))
{
case BackendConstants.Cal_Limit_Extension:
newPaneVM = new LimitUcVM(typeof(LimitUcEditor), outputPath, folder.ParentProject, folder) {
IsDocument = true, IsHidden = true };
break;
case ".xml":
//Old, Do Not Load.
break;
}
break;
case ProjectFileSection.Waveforms:
#region Waveforms
string eDir = MerlinProject.ExtractWaveformSubDirectoryFromFilePath(outputPath);
if (!string.IsNullOrEmpty(eDir)) //if the technology directory exists in merlin test studio.
{
string techDirectory =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Waveforms)
, eDir);
//Create directory if it does not already exist.
if (!Directory.Exists(techDirectory)) { Directory.CreateDirectory(techDirectory); }

//Change the default output path to the correct waveform tech sub directory.
outputPath = Path.Combine(techDirectory, Path.GetFileName(outputPath));
}
```

```
else { } //Else it spits out the waveform in a unspecified technology folder.

//WaveformModel wfm = new WaveformModel() { FilePath = copyOutputPath };
newPaneVM = new
PaneViewModel(typeof(WaveformConverterControls.MainWindowViewModel),
typeof(WaveformConverterControls.WaveformConverterControl), outputPath, folder.ParentProject,
folder)
{
Header = Path.GetFileName(outputPath),
IsDocument = true,
ParentProject = folder.ParentProject,
Icon = "waveform.png",
IsHidden = true
};
switch (Path.GetExtension(inputPath))
{
case ".aiq":
//folder.FolderItems.Add(newPaneVM);
//folder.IsExpanded = true;
//folder.ParentProject.Waveforms.Add(wfm);

////Set the parameters to load the external control with data.
//this.Status = (newPane.ContentUserControl.DataContext as
WaveformConverterControls.MainWindowViewModel).OpenAiqFileCommandExecute(copyOutput
Path);
break;
case ".mtwf2":
if (string.IsNullOrEmpty(eDir)) // Tech Directory null/empty, proceed with normal top level folder.
{
//folder.FolderItems.Add(newPane);
//folder.IsExpanded = true;
//folder.ParentProject.Waveforms.Add(wfm);
}
else
{
//Check if the tree view sub folder needs to be created or not.
ProjectFolder sub_pf = null;
foreach (var sub_item in folder.FolderItems)
{
if (sub_item is ProjectFolder && (sub_item as ProjectFolder).FolderName == eDir)
{
sub_pf = (ProjectFolder)sub_item;
```

```
}
}
if (sub_pf == null)
{
sub_pf = new ProjectFolder() { FolderName = eDir, Section =
ProjectFileSection.WaveformSubFolder, ParentProject = folder.ParentProject };
folder.FolderItems.Add(sub_pf);
}
folder = sub_pf;
//newPaneVM.ParentFolder = sub_pf;
//sub_pf.FolderItems.Add(newPane);
//sub_pf.IsExpanded = true;
////folder.ParentProject.Waveforms.Add(wfm);
}
break;
}
folder.ParentProject.WaveformsChanged();
#endregion Waveforms
break;
case ProjectFileSection.Digital_Patterns:
#region Digital Patterns
switch (Path.GetExtension(inputPath))
{
case ".genericrffepattern":
newPaneVM = new GenericRffePatternVM(typeof(GenericRffePattern), outputPath,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject, IsHidden =
true };
break;
case ".rffepat":
newPaneVM = new DigitalPatternVM(typeof(DigitalPattern), outputPath, folder.ParentProject,
folder) { IsDocument = true, ParentProject = folder.ParentProject, IsHidden = true };
break;
case ".digilevels":
newPaneVM = new DigitalLevelViewModel(typeof(DigitalLevel), outputPath, folder.ParentProject,
folder) { IsDocument = true, ParentProject = folder.ParentProject, IsHidden = true };
break;
case ".digitiming":
newPaneVM = new DigitalTimingVM(typeof(DigitalTiming), outputPath, folder.ParentProject,
folder) { IsDocument = true, ParentProject = folder.ParentProject, IsHidden = true };
break;
}
#endregion Digital Patterns
```

```
break;
case ProjectFileSection.Dll:
switch (Path.GetExtension(inputPath))
{
case ".dll":
newPaneVM = new DllFileInfoVM(typeof(DllFileInfo), outputPath, folder.ParentProject, folder) {
IsDocument = true, IsHidden = true };
break;
}
break;
}

if (dataObj != null)
{
dataObj.FilePath = outputPath;
newPaneVM.RelinkData();
}

if (newPaneVM != null)
{
newPaneVM.IsSaveRequired = true;
newPaneVM.ParentFolder = folder;
folder.FolderItems.Add(newPaneVM);
folder.IsExpanded = true;
}

folder = obj as ProjectFolder; //Reset incase any sub folders needed in addition cycle.
}

//If dialog was shown, Save once at the very end once all existing files are added.
folder.ParentProject.SaveProjectFile(folder.ParentProject.ProjectFilePath);
}

}
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
}
```

```
#endregion

#region Import And Extract Data
public ICommand ImportAndExtractDataCommand { get { return new
DelegateCommand(ImportAndExtractData); } }
private void ImportAndExtractData(object obj)
{
try
{
if (obj is ProjectFolder)
{
ProjectFolder folder = obj as ProjectFolder;

Microsoft.Win32.OpenFileDialog open_file_dlg = new Microsoft.Win32.OpenFileDialog()
{
//No width or height properties???
Multiselect = true
};

//Set the dialog settings and filters.
switch (folder.Section)
{
case ProjectFileSection.Test_Limits:
open_file_dlg.DefaultExt = ".xml";
open_file_dlg.Filter = "XML / CSV Files(.xml; .csv;)|*.xml; *.csv";
open_file_dlg.Title = "Add Existing Test Limit Files";
break;
case ProjectFileSection.Test_Parameters:
break;
case ProjectFileSection.Test_Sequences:
break;
case ProjectFileSection.Calibration_Definitions:
open_file_dlg.DefaultExt = ".xml";
open_file_dlg.Filter = "XML / CSV Files(.xml; .csv;)|*.xml; *.csv";
open_file_dlg.Title = "Import Cal Data V5";
break;
default:
break;
}

if (string.IsNullOrEmpty(open_file_dlg.Filter)) { return; } //No options, Don't show File Dialog.
```

```
//Show dialog and get file path(s).
if (open_file_dlg.ShowDialog() == true)
{
List<string> dialogFileNamesList = new List<string>(open_file_dlg.FileNames);

//iterate through the dialog file names and match with copy output file names.
for (int i = 0; i < dialogFileNamesList.Count(); i++)
{
string inputPath = dialogFileNamesList[i];

MessageBoxResult overwrite_result = MessageBoxResult.None;

PVM newPaneVM = null;

//Create the panes user selected file paths after overwrite check.
switch (folder.Section)
{
case ProjectFileSection.Test_Limits:
//NEEDS testing and shrotening
#region Test Limits PVM
switch (Path.GetExtension(inputPath))
{
case ".csv": //For legacy imports.
case ".xml":
UpperLowerLimitsData limitsData = null;
GoldLimitsData goldsData = null;
OffsetLimitsData offsetsData = null;
string copyOutputPath_Limits =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Test_Limits)
, $"{Path.GetFileNameWithoutExtension(inputPath)}.limits");
string copyOutputPath_Golds =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Test_Limits)
, $"{Path.GetFileNameWithoutExtension(inputPath)}.golds");
string copyOutputPath_Offsets =
Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Test_Limits)
, $"{Path.GetFileNameWithoutExtension(inputPath)}.offsets");

List<string> copyFilePaths = new List<string>() { copyOutputPath_Limits, copyOutputPath_Golds,
copyOutputPath_Offsets };

//Test.LoadAllFromProgramTestLimit(inputPath, out limitsData, out goldsData, out offsetsData);
if (limitsData != null && goldsData != null && offsetsData != null)
```

```
{
foreach(string outputPath in copyFilePaths)
{
#region New Overwrite Check
Action copyLimitsAction = () =>
{
switch (Path.GetExtension(outputPath))
{
case ".limits":
limitsData.SaveToXml(copyOutputPath_Limits); //Save to load in the new VM
break;
case ".golds":
goldsData.SaveToXml(copyOutputPath_Golds); //Save to load in the new VM
break;
case ".offsets":
offsetsData.SaveToXml(copyOutputPath_Offsets); //Save to load in the new VM
break;
}
};
bool wasOverwritePromptedForLimits = OverwriteCheck(folder, inputPath, outputPath,
copyLimitsAction);
if (wasOverwritePromptedForLimits == true)
{
continue;
}
#endregion New Overwrite Check

switch (Path.GetExtension(outputPath))
{
case ".limits":
//Add Test Limits pane.
var newPane_Limits = new TestLimitsVM(typeof(TestLimits), copyOutputPath_Limits,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject, IsHidden =
true };
newPane_Limits.IsSaveRequired = true;
folder.IsExpanded = true;
folder.FolderItems.Add(newPane_Limits);
break;
case ".golds":
//Add Golds pane
var newPane_Golds = new GoldLimitsVM(typeof(GoldLimits), copyOutputPath_Golds,
folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject, IsHidden =
```

```
true };
foreach (GoldLimit glimit in (newPane_Golds as GoldLimitsVM).GoldLimitsObj.Data) //Only do this on import. (Jerry Request).
{
if (glimit.GoldTestLower != 0 || glimit.GoldTestUpper != 0) //If min or max has any data then set to Check_To_Baseline.
{
glimit.GoldRetestControl = GoldControl.Check_To_Baseline;
}
}
newPane_Golds.IsSaveRequired = true;
folder.IsExpanded = true;
folder.FolderItems.Add(newPane_Golds);
break;
case ".offsets":
//Add Offsets pane
var newPane_Offsets = new OffsetLimitsVM(typeof(OffsetLimits), copyOutputPath_Offsets, folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject, IsHidden = true };
newPane_Offsets.IsSaveRequired = true;
folder.IsExpanded = true;
folder.FolderItems.Add(newPane_Offsets);
break;
}
}
}
break;
}
#endregion Test Limits PVM
break;
case ProjectFileSection.Calibration_Definitions:
#region Cal Defs
CalDataModel loadedCdm = null;
string fileNameNoExt = Path.GetFileNameWithoutExtension(inputPath);
string copyOutputPath = Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Project_Data), "Cal Data", $"{fileNameNoExt}{BackendConstants.MTS_Cal_Config_Extension}");

switch (Path.GetExtension(inputPath))
{
case ".xml":
loadedCdm = CalDataModel.LoadFromXmlCalDataV5(inputPath, folder.ParentProject);
```

```
break;
case ".csv":
loadedCdm = CalDataModel.LoadFromCsvCalDataV5(inputPath, folder.ParentProject);
break;
}

#region New Overwrite Check
Action copyAction = () =>
{
loadedCdm.SaveToXml(copyOutputPath);
};
bool wasOverwritePrompted = OverwriteCheck(folder, inputPath, copyOutputPath, copyAction);
if (wasOverwritePrompted == true)
{
continue;
}
#endregion

//Pane Creation.
if (loadedCdm != null)
{
newPaneVM = new CalibrationConfiguratorVM(typeof(CalibrationConfiguratorView),
copyOutputPath, folder.ParentProject, folder) { IsDocument = true, ParentProject =
folder.ParentProject, IsHidden = true };
newPaneVM.IsSaveRequired = true;
newPaneVM.ParentFolder = folder;
folder.FolderItems.Add(newPaneVM);
folder.IsExpanded = true;
}
#endregion
break;
case ProjectFileSection.Calibration_Limits:
//No import option. Add existing.
break;
}

folder = obj as ProjectFolder; //Reset incase any sub folders needed in addition cycle.
}

//If dialog was shown, Save once at the very end once all existing files are added.
folder.ParentProject.SaveProjectFile(folder.ParentProject.ProjectFilePath);
}
```

```
}
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
}
#endregion

//Delete checks for IFileData DataObject before checking DataFile Path !!! bug?
#region Delete File

/// <summary>
/// Renames a current treeView node. Updates the XML and the filesystem.
/// </summary>
public ICommand DeleteFileCommand
{
get
{
return new RelayCommand<object>(DeleteFileCommandExecute, CanDeleteFileExecute);
}
}

/// <summary>
/// Renames a current treeView node. Updates the XML and the filesystem.
/// </summary>
private void DeleteFileCommandExecute(object parameter)
{
try
{
if(parameter is IEnumerable<object>)
{
List<object> selectedItems = new List<object>(parameter as IEnumerable<object>);
foreach(object item in selectedItems)
{
if (item is PVM)
{
PVM paneVM = item as PVM;
switch (paneVM.ParentFolder.Section)
{
```

```
case ProjectFileSection.Test_Parameters:
break;
case ProjectFileSection.Dll:
case ProjectFileSection.Product_Configurations:
case ProjectFileSection.Test_Limits:
case ProjectFileSection.Calibration_Definitions:
case ProjectFileSection.Calibration_Limits:
case Data.ProjectFileSection.Connection_Diagrams:
if (File.Exists(paneVM.DataFilePath))//Remove the actual file if it exists
{
File.Delete(paneVM.DataFilePath);
}
if (Panes.Contains(paneVM)) //If document is open, remove it from host.
{
Panes.Remove(paneVM);
}
paneVM.ParentFolder.FolderItems.Remove(paneVM); //Remove the view model from the project folder.
break;
case ProjectFileSection.WaveformSubFolder:
case ProjectFileSection.Waveforms:
//Remove wrapper model to store file path. Will need to remove once all use cases are gone.
//WaveformModel wfmData = paneVM.DataObject as WaveformModel;
//paneVM.ParentProject.Waveforms.Remove(wfmData);

//var wfFolder = paneVM.ParentProject.ProjectTree.Where(f => f.Section == ProjectFileSection.Waveforms).FirstOrDefault(); //Get top level WF Folder.
//ProjectFile file = wfFolder.FileReferences.Find(x => x.FilePath == wfmData.FilePath); //Get File reference from top level WF folder.
//if (paneVM.ParentFolder.FileReferences.Contains(file))
//{
//    paneVM.ParentFolder.FileReferences.Remove(file); //Remove file reference from parent folder.
//}

if (File.Exists(paneVM.DataFilePath))//Remove the actual file if it exists
{
File.Delete(paneVM.DataFilePath);
}
//Solve region with PVM virtual void Remove()
#region Remove reference usages elsewhere in MTS.
foreach (CalDataModel caldef in paneVM.ParentProject.CalibrationDefinitions) //Remove from
```

```
where it's used before its removed from memory.
{
foreach (CalPointModel cpm in caldef.CalPointModelCollection)
{
if (cpm.Waveform == paneVM.DataFilePath) { cpm.Waveform = string.Empty; }
}
}
#endregion
if (Panes.Contains(paneVM)) //Remove pane from document host collection if it's open.
{
Panes.Remove(paneVM);
}
paneVM.ParentProject.WaveformsChanged();
paneVM.ParentFolder.FolderItems.Remove(paneVM); //Remove pane from project folder.
break;
case ProjectFileSection.Digital_Patterns:
IFileData fileData = paneVM.DataObject as IFileData;
if (paneVM.ParentProject.DigitalPatterns.Contains(fileData))
{
if (File.Exists(fileData.FilePath))  //Deletes the actual file.
{
File.Delete(fileData.FilePath);

//Solve region with PVM virtual void Remove()
///Temp. rffepat generated directory/file removal. May Move later.
if (Path.GetExtension(fileData.FilePath) == BackendConstants.RFFE_Pattern_Extension)
{
DirectoryInfo di = new DirectoryInfo(Path.Combine(Path.GetDirectoryName(fileData.FilePath),
Path.GetFileNameWithoutExtension(fileData.FilePath)));
if (di.Exists)
{
di.Delete(true); //Deletes the directory and all subsequent files. (recursive)
Console.WriteLine($"{fileData.FilePath}: All subsequent generated pattern files have been
deleted.");
}
}
///Temp. rffepat generated directory/file removal. May Move later.
}
paneVM.ParentProject.DigitalPatterns.Remove(fileData); //Removes it from project collection.
}
if (Panes.Contains(paneVM)) { Panes.Remove(paneVM); }
paneVM.ParentFolder.FolderItems.Remove(paneVM);
```

```
break;
default:
//Do nothing for back end folder sections like Project_Data.
break;
}

//Try setting before the pane is deleted, therefore folder items count should equal 1 upon check!
///GUI-BUG: If last item is deleted folder stays expanded (Check to see if this solution works).
if (paneVM.ParentFolder.FolderItems.Count == 0) { paneVM.ParentFolder.IsExpanded = false; }

paneVM.ParentProject.SaveProjectFile(paneVM.ParentProject.ProjectFilePath); //Save file paths
}
else
{
//Is folder or project, do nothing.
}
}

//Need current project to save only once (new property selectedProject in MainWindowVM may fix)
//paneVM.ParentProject.SaveProjectFile(paneVM.ParentProject.ProjectFilePath);
}
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
}
}

/// <summary>
///
/// </summary>
/// <returns>True (always)</returns>
private bool CanDeleteFileExecute(object parameter)
{
return true;
}

#endregion  Delete File

//Make sure it renames the PVM.DataFilePath and IFileData.FilePath object just to make sure
everything is up to date!!!
#region Rename File
```

```
/// <summary>
/// Renames a current treeView node. Updates the XML and the filesystem.
/// </summary>
public ICommand RenameFileCommand
{
get
{
return new RelayCommand<object>(RenameFileCommandExecute,
CanRenameFileCommandExecute);
}
}

/// <summary>
/// Renames a current treeView node. Updates the XML and the filesystem.
/// </summary>
private void RenameFileCommandExecute(object parameter) //NEEDS ALREADY EXISTS OR
OVERWRITE DIALOG
{
try
{
if (parameter is PVM)
{
PVM paneVM = parameter as PVM;
RadWindow.Prompt("Rename", this.OnClosed, ""); //Get input on the new file name.

if (UserDialogEntry != null && !string.IsNullOrEmpty(UserDialogEntry))
{
#region Custom Overwrite check
var existingPane = paneVM.ParentFolder.GetPaneFromHeaderNoExt(UserDialogEntry);
MessageBoxResult overwrite_result = MessageBoxResult.None;
if (existingPane != null)
{
overwrite_result = MessageBox.Show($"{UserDialogEntry} already exists...", "Alert",
MessageBoxButton.OK);
return; //Stop the code if name exists.
}
#endregion Custom Overwrite check

if (!string.IsNullOrEmpty(paneVM.DataFilePath)) //FOR NEW VM CODE
{
string oldDirectory = Path.GetDirectoryName(paneVM.DataFilePath);
```

```
string ext = Path.GetExtension(paneVM.DataFilePath);
string newFileNamePath = Path.Combine(oldDirectory, UserDialogEntry + ext);

File.Move(paneVM.DataFilePath, newFileNamePath); //Renames the file.

//Change internal file data to match external
paneVM.DataFilePath = newFileNamePath;
//paneVM.Header = paneVM.DataFilePath.FileName; //Header should set inside the overridden
property.

paneVM.IsSaveRequired = true;
paneVM.ParentProject.SaveProjectFile(paneVM.ParentProject.ProjectFilePath);
}

}
}
}
catch(Exception ex) { MessageBox.Show(ex.Message); }
}

/// <summary>
///
/// </summary>
/// <returns>True (always)</returns>
private bool CanRenameFileCommandExecute(object parameter)
{
return true;
}

#endregion Rename File

#region Duplicate File
public ICommand DuplicateProjectFileItemCommand { get { return new
DelegateCommand(DuplicateProjectFileItem); } }

private void DuplicateProjectFileItem(object obj)
{
try
{
if(obj != null && obj is PVM)
{
PVM paneToDuplicate = obj as PVM;
```

```
ProjectFolder folder = paneToDuplicate.ParentFolder;
PVM newPane = null;

//Only does cal defs currently.
switch (folder.Section)
{
case ProjectFileSection.Calibration_Definitions:

switch (Path.GetExtension(paneToDuplicate.DataFilePath)) //Continue addition process based on selected file extension.
{
case BackendConstants.MTS_Cal_Config_Extension:
//Create "- copy" file path.
string fileNameNoExt = Path.GetFileNameWithoutExtension(paneToDuplicate.DataFilePath);
string duplicateOutputPath = Path.Combine(folder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Project_Data), "Cal Data", $"{fileNameNoExt} - copy{BackendConstants.MTS_Cal_Config_Extension}");

//Save the selected CalDataModel/PVM to the duplicate output file path.
#region New Overwrite Check
Action copyAction = () =>
{
File.Copy(paneToDuplicate.DataFilePath, duplicateOutputPath, true);
};
bool wasOverwritePrompted = OverwriteCheck(folder, paneToDuplicate.DataFilePath, duplicateOutputPath, copyAction);
if (wasOverwritePrompted == true)
{
return;
}
#endregion

//Create the Pane.
newPane = new CalibrationConfiguratorVM(typeof(CalibrationConfiguratorView), duplicateOutputPath, folder.ParentProject, folder) { IsDocument = true, ParentProject = folder.ParentProject };
break;
}
break;
}

if (newPane != null)
```

```
{
newPane.IsSaveRequired = true;
folder.FolderItems.Add(newPane);
folder.IsExpanded = true;
}

folder.ParentProject.SaveProjectFile(folder.ParentProject.ProjectFilePath);
}
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
}
#endregion

#region Copy Full File Path
/// <summary>
/// Saves to .aiq
/// </summary>
public ICommand CopyFullPathCommand { get { return new DelegateCommand(CopyFullPath); }
}

private void CopyFullPath(object obj)
{
try
{
if(obj != null && obj is PVM)
{
var pvm = (obj as PVM);
Clipboard.SetText(pvm.DataFilePath);
}
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
}
#endregion
```

```
#region Open About Box

/// <summary>
/// Command to open the About Box
/// </summary>
public ICommand OpenAboutBoxCommand
{
get
{
return new RelayCommand(OpenAboutBoxCommandExecute,
CanOpenAboutBoxCommandExecute);
}
}

/// <summary>
/// Command to open the About Box
/// </summary>
private void OpenAboutBoxCommandExecute()
{
try
{
AboutBox about = new AboutBox();
about.Show();
}
catch (Exception ex)
{
trace.TraceError("An error occured : " + ex.Message);
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}
}

/// <summary>
/// Can the command execute.
/// </summary>
/// <returns>True (always)</returns>
private bool CanOpenAboutBoxCommandExecute()
{
return true;
}

#endregion Open About Box
```

```
//Exit command only pertains to the X button top right of the main window...
#region Exit Program

/// <summary>
/// Command to exit the program
/// </summary>
public ICommand ExitProgramCommand
{
get
{
return new RelayCommand(ExitProgramCommandExecute, CanExitProgramCommandExecute);
}
}

/// <summary>
/// Command to exit the program
/// </summary>
public void ExitProgramCommandExecute()
{
try
{
this.trace.TraceInformation("Exit command called!");

// Close Project First and open documents.
bool allClosedSuccessfully = this.CloseProjects(new List<MerlinProject>(this.ProjectsCollection));
//New list avoids collection modified error.

//Save Tool panes for they are all that remain.
this.SaveLayout();

// For the case that the user shuts down the app but files are not saved we prompt them to save
with a dialog box.
// If the user selects cancel then we need to not shut down the application.
if (allClosedSuccessfully == true)
{
System.Windows.Application.Current.Shutdown();
}
}
catch (Exception ex)
{
trace.TraceError("An error occured : " + ex.Message);
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test - Hardware Resource
```

```
Manager");
}
}

/// <summary>
/// Can the command execute.
/// </summary>
/// <returns>True (always)</returns>
private bool CanExitProgramCommandExecute()
{
return true;
}



/// <summary>
/// Closes the current project. Closes all tabs and prompts the user to save the un-saved files.
/// </summary>
/// <returns>True to indicate the project was closed and false to indicate that the user cancelled the close.</returns>
public bool CloseProjects(IEnumerable<MerlinProject> projectsToClose)
{
try
{
List<string> filesThatRequireSaving = new List<string>();
List<PVM> objectsRequiredToBeSaved = new List<PVM>();
foreach (MerlinProject mp in projectsToClose)
{
foreach (ProjectFolder pf in mp.ProjectTree) {
objectsRequiredToBeSaved.AddRange(pf.GetSaveRequiredPanes()); }
}

// Only pop up dialog box if there are files to be saved.
if (objectsRequiredToBeSaved.Count != 0)
{
foreach (PVM pvm in objectsRequiredToBeSaved) // all pvms in this list are IsSaveRequired = true;
{
filesThatRequireSaving.Add(string.Format("{0} > {1} > {2}", pvm.ParentProject.ProjectName, pvm.ParentFolder.FolderName, pvm.Header));
}

// Create an instance of the user control which we can pass to the Diagl Window.
```

```
CloseProjectSaveDiaglog closeProjectSaveProjectDialog = new CloseProjectSaveDiaglog();

var view = new WindowStyled(this)
{

Content = closeProjectSaveProjectDialog,
Title = "Save All Project Files",
WindowState = WindowState.Normal,
SizeToContent = SizeToContent.Manual,
Width = 667,
Height = 375,
ResizeMode = ResizeMode.NoResize,
WindowStartupLocation = WindowStartupLocation.CenterScreen

};

var closeProjDiaglogVM = new CloseProjectSaveDiaglogViewModel(this, view);

closeProjDiaglogVM.FileToSaveList = new ObservableCollection<string>(filesThatRequireSaving);
closeProjDiaglogVM.ObjectsToSave = objectsRequiredToBeSaved;
view.DataContext = closeProjDiaglogVM;

// Display Window.
bool? dialogResult = view.ShowDialog();

if (dialogResult.HasValue && dialogResult.Value == true)
{
foreach(MerlinProject rmp in projectsToClose)
{
//TEMP LOOP TO REMOVE ANY ACTIVE CAL DEF ERRORS ON UNLOAD. Figure out a better
way.
foreach (var calDefModel in rmp.CalibrationDefinitions)
{
foreach(var cpm in calDefModel.CalPointModelCollection)
{
calDefModel.RemoveAllErrors(cpm);
}
} //TEMP LOOP TO REMOVE ANY ACTIVE CAL DEF ERRORS ON UNLOAD. Figure out a better
way.

//Remove any open documents belonging to the projects being removed.
List<PVM> PanesToremove = (Panes.Where(pvm => pvm.IsDocument && pvm.ParentProject !=
```

```
null).Where(pvm => pvm.ParentProject == rmp)).ToList();
foreach (PVM rPvm in PanesToremove) { Panes.Remove(rPvm); }
ProjectsCollection.Remove(rmp);
}
}
else
{
// User Cancelled operation...do nothing.
return false;
}
}
else
{
foreach (MerlinProject rmp in projectsToClose)
{
//TEMP LOOP TO REMOVE ANY ACTIVE CAL DEF ERRORS ON UNLOAD. Figure out a better
way.
foreach (var calDefModel in rmp.CalibrationDefinitions)
{
foreach (var cpm in calDefModel.CalPointModelCollection)
{
calDefModel.RemoveAllErrors(cpm);
}
} //TEMP LOOP TO REMOVE ANY ACTIVE CAL DEF ERRORS ON UNLOAD. Figure out a better
way.

//Remove any open documents belonging to the projects being removed.
List<PVM> PanesToremove = (Panes.Where(pvm => pvm.IsDocument && pvm.ParentProject !=
null).Where(pvm => pvm.ParentProject == rmp)).ToList();
foreach (PVM rPvm in PanesToremove) { Panes.Remove(rPvm); }
ProjectsCollection.Remove(rmp);
}
}

this.Status = "Projects closed successfully.";

}
catch (Exception ex)
{
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}
```

```
return true;
}

#endregion Exit Program

#endregion Commands

//Only has domain unhandled exception method.
#region Private Methods

/// <summary>
/// Method that gets called when an unhandled exception gets thrown. It displays a dialogbox showing the exception. Another showing the inner exception
/// and another showing the stacktrace of the inner exception.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CurrentDomain_UnhandledException(object sender, UnhandledExceptionEventArgs e)
{
try
{
Exception ex = (Exception)e.ExceptionObject;

trace.TraceError("An error occured : " + ex.Message);

MessageBox.Show("Whoops! Please contact the developers with the following"
+ " information:\n\n" + ex.Message + ex.StackTrace,
"Fatal Error", MessageBoxButton.OK, MessageBoxImage.Stop);

if (ex.InnerException != null)
{
MessageBox.Show(ex.InnerException.Message);
MessageBox.Show(ex.InnerException.StackTrace);
}
}
finally
{
//ViewClosing();

//if (RequestClose != null)
//{
```

```
//    RequestClose(null, null);
//}
}
}

#endregion Private Methods

#region Public Methods

//LATEST
public static string GetUniqueFileName(ProjectFolder projectFolder, string NewFileNoExt)
{
int iteration = 1;
string newUniqueName = NewFileNoExt;
while (projectFolder.FolderItems.OfType<PVM>().Any(x =>
Path.GetFileNameWithoutExtension(x.DataFilePath) == newUniqueName))
{
newUniqueName = $"{Path.GetFileNameWithoutExtension(NewFileNoExt)} ({iteration++})";
}
return newUniqueName;
}

/// <summary>
/// Propmts the user if they want to overwrite if the project file in question already exists.
/// Also compensates for existing panes and when using File.Copy() as the overwrite action make
sure overwrite is set to true.
/// </summary>
/// <param name="inputPath"></param>
/// <param name="outputPath"></param>
/// <param name="overwriteAction"></param>
/// <returns>Returns true If the user was prompted about a pre-existing file, otherwise returns
false</returns>
public bool OverwriteCheck(ProjectFolder folder, string inputPath, string outputPath, Action
overwriteAction)
{
#region Overwrite check
var existingPane = folder.GetPaneFromHeader(Path.GetFileName(outputPath));
MessageBoxResult overwrite_result = MessageBoxResult.None;

if (File.Exists(outputPath) && existingPane != null)
{
overwrite_result = MessageBox.Show($"{Path.GetFileName(outputPath)} already exists, Are you
```

```
sure you want to overwrite this file?", "Alert", MessageBoxButton.YesNo);
switch (overwrite_result)
{
case MessageBoxResult.Yes:
//Copy file into project directory with overwrite capability.
if (inputPath != outputPath) //already exists if this equals true, causes error if allowed.
{
overwriteAction.Invoke();

existingPane.LoadData(); //Same File Path. Just Reload and skip pane creation.
existingPane.IsSaveRequired = true;

return true;
}
else
{
Console.WriteLine("Unable to overwrite the same file...");
return true;
}
case MessageBoxResult.No:
return true;
default:
return false; //What is the result when the user clicks the X button on the prompt.
}
}
else if (File.Exists(outputPath) && existingPane == null) //Use case for copying files in the project
folders but not included as a project file yet.
{
return false; //Do nothing and allow the creation panes with already EXISTING project files.
}
else //Use case for normal copy.
{
overwriteAction.Invoke();

return false;
}
#endregion Overwrite check
}



/// <summary>
/// Returns the parent project of the active pane.
```

```
/// </summary>
/// <returns></returns>
public MerlinProject GetActiveParentProject()
{
MerlinProject project = this.GetActivePane().ParentProject;
return project;
}

//Helper Check Method for Project loading
public bool IsProjectNameAlreadyLoaded(string fileName)
{
return this.ProjectsCollection.Where(p => p.ProjectName ==
Path.GetFileNameWithoutExtension(fileName)).Any();
}

/// <summary>
/// The main method for loading projects from throughout the application.
/// </summary>
/// <param name="fIleName"></param>
public void LoadMerlinProject(string fIleName)
{
if (!File.Exists(fIleName))
{
Console.WriteLine("The Project File Chosen to Load No Longer Exists...");
return;
}
if (IsProjectNameAlreadyLoaded(fIleName))
{
Console.WriteLine("The project name you are trying to import is already loaded into the project
tree.");
return;
}

try
{
List<string> missingFiles = new List<string>();

Stopwatch stopwatch = new Stopwatch();
stopwatch.Start();
MerlinProject mp = MerlinProject.LoadFromXML(fIleName, out missingFiles);
stopwatch.Stop();
///MerlinProject mp = await Task.Run(() => MerlinProject.LoadFromXML(fIleName));
```

```
///EXCEPTION: The calling thread must be STA, because many UI components require this.

if (mp != null)
{
this.ProjectsCollection.Add(mp);

///NEED to switch to loading empty PVMs with a "Missing File" Icon.
#region Temp. fix to show all missing file project files at once
if (missingFiles.Count() > 0)
{
string missingMessage = "Missing Project Files: ";
for (int i = 0; i < missingFiles.Count(); i++)
{
missingMessage += $"\n - {missingFiles[i]}";
Console.WriteLine($"{mp.ProjectName} - {missingFiles[i]}");
}
MessageBox.Show(missingMessage);
}
#endregion

MyRecentData.AddRecentProject(mp);//Log as opened recently.
Console.WriteLine("Project File: " + fIleName + $" succesfully loaded (in
{stopwatch.ElapsedMilliseconds} ms).");
}
else { Console.WriteLine("Project object null, cannot add..."); }
}
catch (Exception ex) { Console.WriteLine("File " + fIleName + " failed to load.");
MessageBox.Show(ex.Message);  }
}

//Will need to discontinue support for Legacy .mtp files.
public void LoadLegacyProjectMTP(string FIleName)
{
if (IsProjectNameAlreadyLoaded(FileName))
{
Console.WriteLine("The project name you are trying to import is already loaded into the project
tree.");
return;
}

try
{
```

```
string projectFilePath = FIleName;
string projectSubDirectory = Path.Combine(Path.GetDirectoryName(projectFilePath),
Path.GetFileNameWithoutExtension(projectFilePath));

//string convertedProjectFileName = Path.GetFileNameWithoutExtension(projectFilePath) + "_V2"
+ BackendConstants.MerlinProjectExtension;

//MerlinProject projectBase = new
MerlinProject(Path.Combine(BackendConstants.InitialProjectDirectory,
convertedProjectFileName), this);
MerlinProject projectBase = new MerlinProject(Path.GetDirectoryName(projectFilePath));
projectBase.CreateProjectFileStructure();

XDocument document = XDocument.Load(projectFilePath);
//projectBase.Version = (from myConfig in document.Elements("Project")
//                       select myConfig.Attribute("Version").Value)
//                       .First();
//projectBase.ProjectName = (from myConfig in document.Elements("Project")
//                  select myConfig.Attribute("Name").Value)
//                  .First();
projectBase.ProductName = (from myConfig in document.Elements("Project")
select myConfig.Attribute("Product").Value)
.First();
projectBase.UserTargetSystem = (from myConfig in document.Elements("Project")
select myConfig.Attribute("UserTargetSystem").Value)
.First();
projectBase.TestProgram = (from myConfig in document.Elements("Project")
select myConfig.Attribute("TestProgram").Value).FirstOrDefault();

List<string> calFileRefs = new List<string>();
foreach (XElement fileElement in
document.Element("Project").Element("CalibrationDef").Elements("File"))
{
string name = fileElement.Element("Name").Value;
calFileRefs.Add(Path.Combine(projectSubDirectory, "Calibration Definitions", name));
}
foreach (string calRef in calFileRefs)
{
if (File.Exists(calRef))
{
CalDataModel loadedCdm = CalDataModel.LoadFromJson(calRef);
loadedCdm.ParentProject = projectBase;
```

```
//loadedCdm.FilePath = Path.Combine(projectBase.ProjectDirectory, projectBase.ProjectName,
"Calibration Definitions", Path.GetFileName(calRef));
loadedCdm.FilePath =
Path.Combine(projectBase.GetProjectFileSectionDirectory(ProjectFileSection.Project_Data), "Cal
Data", Path.GetFileName(calRef)); // Set new file path for save.
loadedCdm.GlobalSystemVarient = projectBase.UserTargetSystem;

projectBase.CalibrationDefinitions.Add(loadedCdm);
}
}
string dataPath = Path.Combine(projectSubDirectory, "PinMapData.xml");
if (File.Exists(dataPath))
{
XmlSerializer pinsXmlSerializer = new
XmlSerializer(typeof(ObservableCollection<Data.PinModel>));
using (FileStream pinsFileStream = new FileStream(dataPath, FileMode.Open))
{
projectBase.PinMapDataModel.RfPins =
(ObservableCollection<Data.PinModel>)pinsXmlSerializer.Deserialize(pinsFileStream);
}
}

//this.ProjectsCollection.Add(projectBase);
projectBase.SaveProjectFile(projectBase.ProjectFilePath);
LoadMerlinProject(projectBase.ProjectFilePath);

File.Delete(projectFilePath);

//Log as opened recently.
MyRecentData.AddRecentProject(projectBase);

Console.WriteLine("File " + FIleName + " succesfully loaded.");
}
catch(Exception ex) { Console.WriteLine("File " + FIleName + " failed to load.");
MessageBox.Show(ex.Message); }

}
#endregion Public Methods

#region Pane Layout
public void LoadLayout()
{
```

```
try
{
//Load recent user data.
if (File.Exists(BackendConstants.DockingLayoutFilePath))
{
using (StreamReader writer = new StreamReader(BackendConstants.DockingLayoutFilePath))
{
Type deSerType = typeof(ObservableCollection<PVM>);
XmlSerializer xmlSerializer = new XmlSerializer(deSerType);
var layoutPanes = (ObservableCollection<PVM>)xmlSerializer.Deserialize(writer);
this.Panes.Clear();
foreach (var Lpane in layoutPanes)
{
if (Lpane.IsDocument == false)
{
//Lpane.MVM = this;
this.Panes.Add(Lpane);
}
}
}
}
else
{
this.AddDefaultLayoutPanes();
}
}
catch (Exception ex)
{
MessageBox.Show($"Error occurred trying to load user pane layout.\nContact the developer if this error does not resolve itself upon next application load.\n\nError Message: {ex.Message}");

//Remove old errorous file.
if (File.Exists(BackendConstants.DockingLayoutFilePath))
{
File.Delete(BackendConstants.DockingLayoutFilePath);
}

//Add newest default layout.
AddDefaultLayoutPanes();

Console.WriteLine("Reverted to the default layout.");
}
```

```
}
public void SaveLayout()
{
try
{
using (StreamWriter writer = new StreamWriter(BackendConstants.DockingLayoutFilePath))
{
////Query the master Panes collection to get the non-document panes (tool panes) for serialization as type PaneViewModel to save the layout.
List<PVM> listOfNonDocPanes = (from x in this.Panes where x is PaneViewModel where x.IsDocument == false select (PVM)x).ToList();
ObservableCollection<PVM> nonDocPanesToSerialize = new ObservableCollection<PVM>(listOfNonDocPanes); //Create the expected collection type for next load.

XmlSerializer xmlSerializer = new XmlSerializer(nonDocPanesToSerialize.GetType());
//this.Panes.GetType()
xmlSerializer.Serialize(writer, nonDocPanesToSerialize); //this.Panes
}
}
catch (Exception ex) { MessageBox.Show($"Error occurred trying to save user pane layout => {ex.Message}"); }
}

//Helper Method.
private void AddDefaultLayoutPanes()
{
// default layout in Docking
this.Panes.Add(new PaneViewModel(typeof(ProjectExplorer)) { Header = "Project Explorer", InitialPosition = DockState.DockedLeft });
this.Panes.Add(new PaneViewModel(typeof(SystemExplorer)) { Header = "System Explorer", InitialPosition = DockState.DockedLeft, IsHidden = true });
this.Panes.Add(new PaneViewModel(typeof(CalResultsExplorer)) { Header = "Cal Data Explorer", InitialPosition = DockState.DockedRight, IsHidden = true });
this.Panes.Add(new PaneViewModel(typeof(Output)) { Header = "Output", InitialPosition = DockState.DockedBottom });
this.Panes.Add(new PaneViewModel(typeof(ErrorList)) { Header = "Error List", InitialPosition = DockState.DockedBottom, IsHidden = true });
this.Panes.Add(new PaneViewModel(typeof(PropertiesPane)) { Header = "Properties", InitialPosition = DockState.DockedLeft, IsHidden = true });
}
#endregion Pane Layout
```

```
#region Docking Pane Methods

//Opens an already opened document with a single click.

#region ProjectTreeViewItemSingleClickedCommand
public ICommand ProjectTreeViewItemSingleClickedCommand { get { return new
DelegateCommand(ProjectTreeViewItemSingleClicked); } }
private void ProjectTreeViewItemSingleClicked(object args)
{
RadTreeViewItem item = (RadTreeViewItem)(args as RadRoutedEventArgs).OriginalSource;
if (item.DataContext is PVM)
{
PVM paneViewModel = (PVM)item.DataContext;
if (this.Panes.Contains(paneViewModel))
{
foreach(PVM pvm in this.Panes) { pvm.IsActive = false; } //Set all other panes to false to avoid a
auto pane selection bug.
paneViewModel.IsActive = true;
}
}
}
#endregion

#region ProjectTreeViewItemDoubleClickedCommand
//Opens a hidden document
public ICommand ProjectTreeViewItemDoubleClickedCommand { get { return new
DelegateCommand(ProjectTreeViewItemDoubleClicked); } }
private void ProjectTreeViewItemDoubleClicked(object args)
{
RadTreeViewItem item = (RadTreeViewItem)(args as RadRoutedEventArgs).OriginalSource;

if(item.DataContext is PaneViewModel)
{
CreateNewDocument(item.DataContext as PaneViewModel);
}
if(item.DataContext is PVM) //Temp Code
{
var paneViewModel = item.DataContext as PVM;
if (!this.Panes.Contains(paneViewModel))
{
paneViewModel.IsHidden = false;
```

```
this.Panes.Add(paneViewModel);
}
else
{
paneViewModel.IsActive = true;
}
}
}
#endregion

#region Cal Results stuff
public ICommand ImportResultsCommand { get { return new DelegateCommand(ImportResults); }
}
private void ImportResults(object obj)
{
try
{
// Create OpenFileDialog
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog()
{
DefaultExt = ".xml",
Filter = "XML File (.xml)|*.xml",
Title = "Open"
};
Nullable<bool> result = dlg.ShowDialog();

if (result == true)
{
var calData = CalDataV5.ImportCalDataXML(dlg.FileName);

if (calData.ResultsData == false) //Config file, don't allow import.
{
MessageBox.Show("Calibration Definiton file invalid!(Calibration Configuration File format)");
}
else //Results file, allow import.
{
///SHOULD be converted to a dialog on whether or not the user would like to replace the current result or cancel.
//Chech to see if the Cal Data file name hasn't been added already. (Bug: Active tab selection is currently uses headers to manage panes).
string cdmFileName = Path.GetFileNameWithoutExtension(dlg.FileName);
foreach (CalDataModel cdm in CalibrationResultsCollection)
```

```
{
if (cdm.FileName == cdmFileName)
{
string msg = $"{cdmFileName} has already been added. Please remove it and try again.";
MessageBox.Show(msg);
Console.WriteLine(msg);
return;
}
}

var cdmToAdd = CalDataModel.CreateCalDataModelsFromCalDataV5(ref calData, dlg.FileName);
CalibrationResultsCollection.Add(cdmToAdd);
}
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

public ICommand CreateCalLimitsModelCommand { get { return new
DelegateCommand(CreateCalLimitsModel); } }
private void CreateCalLimitsModel(object obj)
{
if(obj != null)
{
CalDataModel cdm = obj as CalDataModel;

///Build a project file assignment window !!!
//User Assigned project to store the Cal Limits.
#region Project Assignment
MerlinProject project;
if(this.ProjectsCollection.Count == 1)
{
project = this.ProjectsCollection.FirstOrDefault();
}
else if (this.ProjectsCollection.Count > 1)
{
//Replace with the window to choose which project you want to send the cal limits model too. OR
selected project option (could extrapolate)
MessageBox.Show("There are multiple projects loaded. Placing Limits file in the first project.
Window to choose which project will be available soon. "); //Remove.
project = this.ProjectsCollection.FirstOrDefault(); //Remove.
}
```

```
else //Must be 0 projects loaded.
{
MessageBox.Show("There are no projects loaded to accept the limits file. Would you like to load one in?");
//Provide the open project window. Once loaded add the file.
return;
}
#endregion Project Assignment

if (project != null)
{
#region Create the File Path
string fileNameOfCdmNoExtension = Path.GetFileNameWithoutExtension(cdm.FilePath);
ProjectFolder CalLimitsFolder = project.GetProjectFolder(ProjectFileSection.Calibration_Limits);

//string uniqueFileName = MainWindowViewModel.GetUniqueFileName(CalLimitsFolder, fileNameOfCdmNoExtension);
//string filePath = Path.Combine(CalLimitsFolder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Calibration_Limits), $"{uniqueFileName}{BackendConstants.Cal_Limit_Extension}");

string filePath = Path.Combine(CalLimitsFolder.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Calibration_Limits), $"{fileNameOfCdmNoExtension}{BackendConstants.Cal_Limit_Extension}");
#endregion

#region Check IsLimitFailed Prompt
MessageBoxResult ContinueWithIsLimitFailed_result = MessageBoxResult.None;
if (cdm.IsLimitFailed == true)
{
ContinueWithIsLimitFailed_result = MessageBox.Show($"{Path.GetFileName(filePath)} contains failed limit(s), Click OK to continue creating Calibration Limits", "Alert", MessageBoxButton.OKCancel);
switch (ContinueWithIsLimitFailed_result)
{
case MessageBoxResult.OK:
//Do nothing and continue the code.
break;
case MessageBoxResult.Cancel:
return; //Stop the code
}
}
```

```
#endregion Check IsLimitFailed Prompt

#region Create the Cal Limits Model and Internal Expando Objects

CalLimitsModel limitsModel = new CalLimitsModel(filePath, cdm) { CalibrationDataVersion =
cdm.CalibrationDataVersion };

foreach (CalResult calresult in cdm.Results)
{
if (BackendConstants.CalLimit_Producible_Results.Contains(calresult.ResultName))
{
string newCalResultLimitsName = calresult.ResultName.Replace("Results", string.Empty);
//Removes the word results.

//Determine whether the limits are for CalFactor or ENR.
string LimitProperty = ""; //represents whether the results are of CalFactor or ENR.
object resultSample = calresult.KvPMergedObjectData.FirstOrDefault();
foreach (PropertyInfo prop in resultSample.GetType().GetProperties())
{
if (prop.Name == "CalFactor") { LimitProperty = "CalFactor"; }
if (prop.Name == "ENR") { LimitProperty = "ENR"; }
}

//Create the ExpandoObjects using the objects from the CalResult, then replace the data.
List<dynamic> newMergedObjData = new List<dynamic>();
foreach (dynamic mergedObj in calresult.KvPMergedObjectData)
{
var exObj = new ExpandoObject();
var dictionary = (IDictionary<string, object>)exObj;

int passOne = 0;
int lastKeyIndexWhenValueForKeyInference = 0;
var listOfProperties = mergedObj.GetType().GetProperties();
bool canAddLimitItem = true;
foreach (var property in listOfProperties)
{
var kvpRef = calresult.KeyOrValuePropertyReferences.Find(x => x.Value == property.Name);
//Not used in 'property.Name == LimitProperty' case.
if (property.Name == LimitProperty)
{
//Get Baseline for use below.
double Baseline = (double)property.GetValue(mergedObj); //Get the baseline value for MinMax
```

```
calculations.
Baseline = Math.Round(Baseline, 3); //Rounds to the 3rd decimal place (to avoid unwanted long UI numbers).

//double powerdBmDeltaLimit = 0.2;
double tempMinTolerance = CalLimitsModel.DefaultMinTolerance;
double tempMaxTolerance = CalLimitsModel.DefaultMaxTolerance;

//Check ResultPass to determine NaN Limits (Outer Loop)
var resultPassProp = mergedObj.GetType().GetProperty("ResultPass");
if (resultPassProp != null)
{
bool resultPassToCheck = (bool)resultPassProp.GetValue(mergedObj);
if (resultPassToCheck == false) //Failed. Set NaN Tolerances.
{
tempMinTolerance = double.NaN;
tempMaxTolerance = double.NaN;
}
else //Passed. Set to default tolerances.
{
var powerdBmDeltaProp = mergedObj.GetType().GetProperty("PowerdBmDelta");
if (powerdBmDeltaProp != null)
{
double powerdBmDelta = (double)powerdBmDeltaProp.GetValue(mergedObj);
if (powerdBmDelta <= 0.2 && powerdBmDelta >= -0.2) //If within these delta limits, Set NaN tolerances.
{
tempMinTolerance = double.NaN;
tempMaxTolerance = double.NaN;
}
else //Passed & Delta Is not within min range. Use default tolerances.
{
tempMinTolerance = CalLimitsModel.DefaultMinTolerance;
tempMaxTolerance = CalLimitsModel.DefaultMaxTolerance;
}
}
//tempMinTolerance = CalLimitsModel.DefaultMinTolerance;
//tempMaxTolerance = CalLimitsModel.DefaultMaxTolerance;
}
}

// Check if the WasTargetLevelAchieved property exists
```

```
var wasTargetLevelAchievedProp =
mergedObj.GetType().GetProperty("WasTargetLevelAcheived");
var amplifiersCompressedProp = mergedObj.GetType().GetProperty("AmplifiersCompressed");
if (wasTargetLevelAchievedProp != null)
{
bool wasTargetLevelAchieved = (bool)wasTargetLevelAchievedProp.GetValue(mergedObj);
if (wasTargetLevelAchieved == false)
{
// if WasTargetLevelAchieved is false, set Min, max, tolerances to double.NaN
dictionary.Add("MinTolerance", double.NaN);
dictionary.Add("Min", double.NaN);
dictionary.Add("Baseline", Baseline);
dictionary.Add("Max", double.NaN);
dictionary.Add("MaxTolerance", double.NaN);
}
else
{
dictionary.Add("MinTolerance", tempMinTolerance);
double min = Math.Round(Baseline - tempMinTolerance, 3);
dictionary.Add("Min", min);
dictionary.Add("Baseline", Baseline);
double max = Math.Round(Baseline + tempMinTolerance, 3);
dictionary.Add("Max", max);
dictionary.Add("MaxTolerance", tempMinTolerance);
}
}
else if (amplifiersCompressedProp != null)
{
bool amplifiersCompressed = (bool)amplifiersCompressedProp.GetValue(mergedObj);
if (amplifiersCompressed == true)
{
// if WasTargetLevelAchieved is false, set Min, max, tolerances to double.NaN
dictionary.Add("MinTolerance", double.NaN);
dictionary.Add("Min", double.NaN);
dictionary.Add("Baseline", Baseline);
dictionary.Add("Max", double.NaN);
dictionary.Add("MaxTolerance", double.NaN);
}
else
{
dictionary.Add("MinTolerance", tempMinTolerance);
double min = Math.Round(Baseline - tempMinTolerance, 3);
```

```
dictionary.Add("Min", min);
dictionary.Add("Baseline", Baseline);
double max = Math.Round(Baseline + tempMinTolerance, 3);
dictionary.Add("Max", max);
dictionary.Add("MaxTolerance", tempMinTolerance);
}
}
else
{
dictionary.Add("MinTolerance", tempMinTolerance);
double min = Math.Round(Baseline - tempMinTolerance, 3);
dictionary.Add("Min", min);
dictionary.Add("Baseline", Baseline);
double max = Math.Round(Baseline + tempMinTolerance, 3);
dictionary.Add("Max", max);
dictionary.Add("MaxTolerance", tempMinTolerance);
}
}
else
{
//Property Name (value) to KeyOrValue Reference to make sure all key properties are generated.
//var kvpRef = calresult.KeyOrValuePropertyReferences.Find(x => x.Value == property.Name);
if (kvpRef.Key == "Key") //Only Add if it's a Key property, we do not need any other Value
properties.
{
dictionary.Add(property.Name, property.GetValue(mergedObj));
lastKeyIndexWhenValueForKeyInference++;
}

////Check ResultPass if it exists to determine limit item addition to CalLimits.
//if (kvpRef.Key == "Value" && kvpRef.Value == "ResultPass")
//{
//    var resultPassToCheck = property.GetValue(mergedObj);
//    if ((bool)resultPassToCheck == false)
//    {
//        Console.WriteLine("Rejected a limit: ResultPass == false...");
//        canAddLimitItem = false; //Skip so object is not added to limits.
//    }
//}

//WasHfPathUsed Inference from ifFreq property value being NaN.
if (newCalResultLimitsName == "measRf210HfPortAmpConfigurations" && passOne == 0 &&
```

```
property.Name == "ifFreq")
{
var ifFreqToCheck = property.GetValue(mergedObj);
if (double.IsNaN((double)ifFreqToCheck))
{
dictionary.Add("WasHfPathUsed", (bool)false);
calresult.KeyOrValuePropertyReferences.Insert(lastKeyIndexWhenValueForKeyInference, new
KvP<string, string>("Key", "WasHfPathUsed"));
//passOne++;
}
else
{
dictionary.Add("WasHfPathUsed", (bool)true);
calresult.KeyOrValuePropertyReferences.Insert(lastKeyIndexWhenValueForKeyInference, new
KvP<string, string>("Key", "WasHfPathUsed"));
//passOne++;
}
passOne++;
}

}

}
if(canAddLimitItem == true)
{
newMergedObjData.Add(exObj);
}
}

CalResult newResult = new CalResult() //Create a new result as to not affect the viewer data.
Clone.
{
ResultName = newCalResultLimitsName,
Parent = calresult.Parent,
CalDataModelParent = calresult.CalDataModelParent,

KeyName = calresult.KeyName,
ValueName = calresult.ValueName,
KeyOrValuePropertyReferences = calresult.KeyOrValuePropertyReferences,
KvPMergedObjectData = newMergedObjData
};
```

```
limitsModel.Results.Add(newResult);
}
}

////limitsModel.Save_Custom_XML(filePath); //TEMP
//limitsModel.SaveToXml(filePath);
#endregion

#region Cal Def Hash set via file name mathcing
string cdmFileNameNoExt = Path.GetFileNameWithoutExtension(filePath);
string cdmFilePath =
Path.Combine(project.GetProjectFileSectionDirectory(ProjectFileSection.Calibration_Definitions),
$"{cdmFileNameNoExt}.xml");
if (File.Exists(cdmFilePath))
{
try
{
var originalCalDataForHashSet = CalDataV5.ImportCalDataXML(cdmFilePath);
limitsModel.HashCalDef = originalCalDataForHashSet.Hash;
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
return; //Stop the code.
}
}
else
{
MessageBox.Show($"Cal Def {cdmFileNameNoExt} not found in project: Unable to set
HashCalDef. Cal Limit Creation Cancelled.");
return; //Stop the code.
}
#endregion

#region Overwrite check
MessageBoxResult overwrite_result = MessageBoxResult.None;
var existingPane = CalLimitsFolder.GetPaneFromHeader(Path.GetFileName(filePath));
if (File.Exists(filePath) && existingPane != null)
{
overwrite_result = MessageBox.Show($"{Path.GetFileName(filePath)} already exists, Are you sure
you want to overwrite this file?", "Alert", MessageBoxButton.YesNo);
switch (overwrite_result)
```

```
{
case MessageBoxResult.Yes:

//Copy file into project directory with overwrite capability.
if (cdm.FilePath != filePath) //already exists if this equals true, causes error if allowed.
{
limitsModel.SaveToXml(filePath);
existingPane.LoadData();
existingPane.IsSaveRequired = true;
return; //Skip this to not produce a new PVM.
}
else
{
Console.WriteLine("Unable to overwrite the same file...");
return; //Skip this to not produce a new PVM.
}
break;
case MessageBoxResult.No:
return; //Skip this to not produce a new PVM.
}
}
else if (File.Exists(filePath) && existingPane == null) //Use case for copying files in the project
folders but not included as a project file yet.
{
//Do nothing and allow the creation panes with already EXISTING project files.
}
else
{
limitsModel.SaveToXml(filePath);
}
#endregion Overwrite check

#region PVM production and Selection
string header = limitsModel.FileName;
PVM pane = null;
if (CheckPaneHeaderExists(header, out pane) != true)
{
pane = new LimitUcVM(typeof(LimitUcEditor), filePath, CalLimitsFolder.ParentProject,
CalLimitsFolder) { InitialPosition = DockState.DockedLeft, IsDocument = true };
CreateNewDocument(pane);
}
else { pane.IsActive = true; } //Select Pane.
```

```
//Make sure everything is added to the correct collections and project folder.
pane.IsSaveRequired = true;
CalLimitsFolder.FolderItems.Add(pane);
CalLimitsFolder.IsExpanded = true;
CalLimitsFolder.ParentProject.CalibrationLimits.Add(limitsModel);
project.SaveProjectFile(project.ProjectFilePath);
#endregion PVM production and Selection
}
}
}

public ICommand RemoveCalResultCommand { get { return new
DelegateCommand(RemoveCalResult); } }
private void RemoveCalResult(object obj)
{
CalDataModel model = (obj as CalDataModel);
if (!string.IsNullOrEmpty(model?.FileName))
{
List<CalDataModel> itemsToRemove = CalibrationResultsCollection.Where(item =>
item.FileName == model.FileName).ToList(); //May want to filter on FilePath rather than FileName.
foreach (var mr in itemsToRemove)
{
//Remove all open result panes with matching parent Cal Data.
List<PVM> PanesToremove = new List<PVM>();
PanesToremove.AddRange((Panes.Where(pvm => pvm is ResultDataVM).Where(pvm => (pvm
as ResultDataVM).CalDataInfo == model)).ToList());
PanesToremove.AddRange((Panes.Where(pvm => pvm is
AttenuatorResultsViewerVM).Where(pvm => (pvm as AttenuatorResultsViewerVM).CalDataInfo
== model)).ToList());
foreach (PVM rPvm in PanesToremove)
{
Panes.Remove(rPvm);
}

//Remove Cal Data from results collection.
CalibrationResultsCollection.Remove(mr);
}
}
}

public ICommand ResultTreeViewItemDoubleClickedCommand { get { return new
```

```
DelegateCommand(ResultTreeViewItemDoubleClicked); } }
private void ResultTreeViewItemDoubleClicked(object args)
{
RadTreeViewItem item = (RadTreeViewItem)(args as RadRoutedEventArgs).OriginalSource;

if (item.DataContext != null)
{
if(item.DataContext is CalDataModel)
{
CalDataModel selectedModel = (item.DataContext as CalDataModel);
string header = selectedModel.FileName;
PVM pane = null;
if (CheckPaneHeaderExists(header, out pane) != true)
{
pane = new ResultDataVM(typeof(CalDataInfo), selectedModel) { Header = header, InitialPosition
= DockState.DockedLeft, IsDocument = true };
CreateNewDocument(pane);
//pane.ContentUserControl.DataContext = selectedModel;
}
else { pane.IsActive = true; } //Select Pane.
}
else if (item.DataContext is CalResult)
{
CalResult selectedResult = (item.DataContext as CalResult);
string header = $"{selectedResult.Parent.FileName} - {selectedResult.ResultName}";
PVM pane = null;
if (CheckPaneHeaderExists(header, out pane) != true)
{
switch (selectedResult.ResultName)
{
case "srcPortRF110CourseGrainAttenuatorResults":
case "srcPortRF110FineGrainAttenuatorResults":
pane = new AttenuatorResultsViewerVM(typeof(AttenuatorResultsViewer), selectedResult,
selectedResult.Parent) { Header = header, InitialPosition = DockState.DockedLeft, IsDocument =
true };
CreateNewDocument(pane);
//pane.ContentUserControl.DataContext = new AttenuatorResultsViewerVM(selectedResult);
break;
default:
pane = new ResultDataVM(typeof(ResultDataView), selectedResult, selectedResult.Parent) {
Header = header, InitialPosition = DockState.DockedLeft, IsDocument = true };
CreateNewDocument(pane);
```

```
//pane.ContentUserControl.DataContext = new ResultDataVM(selectedResult);
break;
}
}
else { pane.IsActive = true; } //Select Pane.
}
}
}
#endregion

#region ShowSystemConfigCommand
public ICommand ShowSystemConfigCommand { get { return new
DelegateCommand(ShowSystemConfig); } }
private void ShowSystemConfig(object obj)
{
string header = "System Configuration";
PVM pane = null;
if(CheckPaneHeaderExists(header, out pane) != true)
{
var systemConfigPane = new PaneViewModel(typeof(SystemConfiguration)) { Header = header,
InitialPosition = DockState.DockedLeft, IsDocument = true };
CreateNewDocument(systemConfigPane);
}
else { pane.IsActive = true; } //Select Pane.
}
#endregion

/// <summary>
/// Checks if the pane header provided exists in the active/open panes collection.
/// </summary>
/// <param name="paneHeader"></param>
/// <param name="pane"></param>
/// <returns></returns>
public bool CheckPaneHeaderExists(string paneHeader, out PVM pane)
{
bool exists = false;
pane = null;
foreach (PVM pvm in from PVM pvm in this.Panes
where pvm.Header == paneHeader
select pvm)
{
exists = true;
```

```
pane = pvm;
}
return exists;
}

/// <summary>
/// Returns the active/currently selected pane.
/// </summary>
/// <returns></returns>
public PVM GetActivePane()
{
PVM activePvm = null;
foreach (var pvm in this.Panes.Where(pvm => pvm.IsActive))
{
activePvm = pvm;
}
return activePvm;
}

/// <summary>
/// Adds, or if it exists, then sets active the pane passed in as the PVM parameter.
/// </summary>
/// <param name="paneViewModel"></param>
public void CreateNewDocument(PVM paneViewModel)
{
if (!this.Panes.Contains(paneViewModel))
{
paneViewModel.IsHidden = false;
this.Panes.Add(paneViewModel);
}
else
{
paneViewModel.IsActive = true;
}
}


///Experimental REIGION NOT FOR PRODUCTION.
#region PVM Factory
public static PVM GeneratePane(IFileData dataModel)
{
PVM pane = null;
```

```
//return a pane with a view respective to the model type.

return pane;
}
public static PVM GeneratePane(MerlinProject project)
{
PVM pane = null;

//return a pane with a view respective to the merlin project model type.

return pane;
}
#endregion
///Experimental REIGION NOT FOR PRODUCTION.

#endregion

public void OnClosed(object sender, WindowClosedEventArgs e)
{
UserDialogEntry = null;
if (e.PromptResult != null)
{
UserDialogEntry = e.PromptResult.ToString();
}
}
}
}
```

PaneViewModel.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.UserControls;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Controls;
using System.Xml.Serialization;
using Telerik.Windows.Controls;
```

```
using Telerik.Windows.Controls.Docking;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public interface IPVMLink
{
PaneViewModel PVM { get; set; }
}

//This interface to eventually be phased out.
public interface IFileData
{
string FileName { get; }
string FilePath { get; set; }
//Type ModelType { get; }

event Action CommitEditBeforeSave;
event Action ChangeOccured;

void Save();

void Load();

//Returns true or false whether or not a operation was completed.
bool Undo();

//Returns true or false whether or not a operation was completed.
bool Redo();
}

//To be removed or used as tool pane VM?
[XmlInclude(typeof(PaneViewModel))]
[Serializable]
public class PaneViewModel : PVM
{
#region Private Members
private string header;
private DockState initialPosition;
private bool isActive;
private bool isHidden;
private bool isDocument;
private Type _contentType;
```

```
private bool _isSaveRequired = false;
private MerlinProject _parentProject;
#endregion

#region Constructors
public override void PaneClose()
{
if(DataContextType != null && ContentType != null) //For waveforms only...
{
this.SaveOccurred -=
((WaveformConverterControls.MainWindowViewModel)this.ViewDataContext).SaveOccuredSubsc
riber;
((WaveformConverterControls.MainWindowViewModel)this.ViewDataContext).ChangeOccured -=
this.ChangeOccuredSubscriber;

this.ContentUserControl = null;
this.ViewDataContext = null;
this.ViewDataContext = Activator.CreateInstance(DataContextType);
}
//GC.Collect();
//this.View = null; //There is a user control stored but this is supported legacy functionality. Do NOT
Nullify.
}

public PaneViewModel() { } //De-serialization Constructor.
public PaneViewModel(Type contentType)  : base(contentType) //Tool Constructor. (Side Panes)
{
this.ContentType = contentType;
}
public PaneViewModel(Type contentType, MerlinProject parentProject) : base(contentType,
parentProject)  //Static Document Constructor.
{
//this.ParentProject = parentProject;
this.ContentType = contentType;
}
public PaneViewModel(Type contentType, IFileData dataObject, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataObject, parentProject, parentFolder)
//Document Constructor.
{
//this.DataObject = dataObject;
//this.ParentProject = parentProject;
```

```
this.ContentType = contentType;
//this.ParentFolder = parentFolder;

if(dataObject != null) { this.DataObject.ChangeOccured += ChangeOccuredSubscriber; }
}
public PaneViewModel(Type dataContextType, Type contentType, string dataFilePath,
MerlinProject parentProject, ProjectFolder parentFolder) : base(contentType, dataFilePath,
parentProject, parentFolder)  //Document Constructor.
{
//this.DataObject = dataObject;
//this.ParentProject = parentProject;
this.DataContextType = dataContextType;
this.ContentType = contentType;
//this.ParentFolder = parentFolder;

//if (dataObject != null) { this.DataObject.ChangeOccured += ChangeOccuredSubscriber; }
}

~PaneViewModel()
{
//this.DataObject.ChangeOccured -= ChangeOccuredSubscriber;
}

public void ChangeOccuredSubscriber() { IsSaveRequired = true; }
#endregion

#region PaneViewModel Attributes

[XmlIgnore]
public Type ContentType
{
get { return _contentType; }
private set
{
_contentType = value;
OnPropertyChanged("ContentType");
if (ParentProject != null)
{
if(DataContextType != null) //Temp If statement for waveforms.
{
//Do not load the waveforms control.
}
```

```
else //Regular PaneViewModel usage.
{
ContentUserControl = Activator.CreateInstance(value) as UserControl;
//View/ViewModel dependencies need to be handled some other way  so the user control can spool with null data context (ex: usercontrol: IView => viewModel.View {get set})
//Then assign the DataContext ViewModel with PVM as abstract base class (and overrides for save, undo, etc.) WHEN => only when an instance needs to be created!!!

if ((ContentUserControl as UserControl).DataContext is IPVMLink)
{
((ContentUserControl as UserControl).DataContext as IPVMLink).PVM = this;
}
}

}
}
}

[XmlAttribute]
public string TypeName
{
//this.ContentType.AsseblyQualifiedName;
get { return this.ContentType == null ? string.Empty : this.ContentType.FullName; }
set { this.ContentType = value == null ? null : Type.GetType(value); }
}

private Type _dataContextType;
[XmlIgnore]
public Type DataContextType
{
get { return _dataContextType; }
private set
{
_dataContextType = value;
OnPropertyChanged("DataContextType");
if (ParentProject != null)
{
ViewDataContext = Activator.CreateInstance(value);
}
}
}
#endregion
```

```
#region MTS Modifier Attributes

private UserControl _contentUserControl;
[XmlIgnore]
public UserControl ContentUserControl
{
get { return _contentUserControl;  }
set {
_contentUserControl = value;
OnPropertyChanged("ContentUserControl");
}
}

private object _viewDataContext;
[XmlIgnore]
public object ViewDataContext
{
get { return _viewDataContext; }
set { _viewDataContext = value; OnPropertyChanged("ViewDataContext"); }
}
#endregion

}

////Test Case Pane.
//[XmlInclude(typeof(MainToolViewModel))]
//[Serializable]
//public class MainToolViewModel : PVM
//{
//    public override void PaneClose()
//    {
//        //No View to nullify.
//    }
//}



//[XmlInclude(typeof(MainToolViewModel))]
[Serializable]
[XmlInclude(typeof(PaneViewModel))]
public abstract class PVM : ViewModelBase
```

```
{
#region Private Members
private string header;
private DockState initialPosition;
private bool isActive;
private bool isHidden;
private bool isDocument;
private Type _contentType;

private bool _isSaveRequired = false;
private MerlinProject _parentProject;
#endregion

public virtual string DataFilePath
{
get; set;
}

//Set in the child class.
[XmlIgnore]
public virtual List<ContextMenuItem> CMSource { get; set; }

#region Constructors
public PVM() { } //De-serialization Constructor.
public PVM(Type contentType) //Tool Constructor. (Side Panes)
{
this.ContentType = contentType;
}
public PVM(Type contentType, MerlinProject parentProject) //Static Document Constructor.
{
this.ParentProject = parentProject;
this.ContentType = contentType;
}
public PVM(Type contentType, IFileData dataObject, MerlinProject parentProject, ProjectFolder
parentFolder) //Document Constructor.
{
this.DataObject = dataObject;
this.ParentProject = parentProject;
this.ContentType = contentType;
this.ParentFolder = parentFolder;

if(this.DataObject != null)
```

```
{
this.DataObject.ChangeOccured += ChangeOccuredSubscriber;
}
}
public PVM(Type contentType, string dataFilePath, MerlinProject parentProject, ProjectFolder
parentFolder) //NEW DOCUMENT Constructor.
{
this.DataFilePath = dataFilePath;
this.ParentProject = parentProject;
this.ContentType = contentType;
this.ParentFolder = parentFolder;

}

~PVM() //Might not work.
{
//this.DataObject.ChangeOccured -= ChangeOccuredSubscriber;
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public event Action SaveOccurred;

public virtual void SaveData()
{
if (DataObject != null) //Supports the old method of saving
{
DataObject.Save();
}
SaveOccurred?.Invoke();
this.IsSaveRequired = false;
}

public virtual void LoadData()
{
//Override loading code goes above base.LoadData();

this.RelinkData(); //Relink Data after load.
this.ValidateData(); //Then Check for any data validation errors.
}

public virtual void Undo()
{
```

```
//used in Cal Def and Pin Map
}

public virtual void Redo()
{
//used in Cal Def and Pin Map
}

public virtual void Duplicate()
{

}
public virtual void Copy()
{

}
public virtual void Paste()
{

}

public virtual void ValidateData()
{
//Standard.
}

public virtual void RelinkData()
{
//Standard.
}
#endregion

#region PaneViewModel Attributes
[XmlAttribute]
public string Header
{
get { return this.header; }
set
{
if (this.header != value)
{
this.header = value;
```

```
this.OnPropertyChanged("Header");
}
}
}

[XmlAttribute]
public DockState InitialPosition
{
get { return this.initialPosition; }
set
{
if (this.initialPosition != value)
{
this.initialPosition = value;
this.OnPropertyChanged("InitialPosition");
}
}
}

[XmlIgnore] //[XmlAttribute]
public bool IsActive
{
get
{
return this.isActive;
}
set
{
if (this.isActive != value)
{
this.isActive = value;
this.OnPropertyChanged("IsActive");
}
}
}

[XmlAttribute]
public bool IsHidden
{
get
{
return this.isHidden;
```

```
}
set
{
if (this.isHidden != value)
{
this.isHidden = value;
this.OnPropertyChanged("IsHidden");
}
}
}
public abstract void PaneClose();

[XmlAttribute]
public bool IsDocument
{
get
{
return this.isDocument;
}
set
{
if (this.isDocument != value)
{
this.isDocument = value;
this.OnPropertyChanged("IsDocument");
}
}
}


[XmlIgnore]
public Type ContentType
{
get { return _contentType; }
private set
{
_contentType = value;
OnPropertyChanged("ContentType");
}
}

[XmlAttribute]
```

```
public string TypeName
{
//this.ContentType.AsseblyQualifiedName;
get { return this.ContentType == null ? string.Empty : this.ContentType.FullName; }
set { this.ContentType = value == null ? null : Type.GetType(value); }
}
#endregion

#region MTS Modifier Attributes
[XmlAttribute]
public bool IsStatic { get; set; }

[XmlIgnore]
public bool IsSaveRequired
{
get { return _isSaveRequired; }
set { _isSaveRequired = value; OnPropertyChanged("IsSaveRequired"); }
}

//Create dynamic icon selector later.
private string _icon = "/Images/Document_16x.png";
public string Icon
{
get { return _icon; }
set { _icon = value; OnPropertyChanged("Icon"); }
}

[XmlIgnore]
public IFileData DataObject { get; set; } //private set? cannot be changed during runtime, would cause binding issues with a change in user controls.

//[XmlIgnore]
//public bool IsDataLoaded { get { return DataObject != null ? true : false; } } //If data object is not null, then there is data loaded, whether or not it is the right data is implied upon access/use.

[XmlIgnore]
public MerlinProject ParentProject
{
get { return _parentProject; }
set
{
_parentProject = value;
```

```
OnPropertyChanged("ParentProject");
}
}
[XmlIgnore]
public ProjectFolder ParentFolder { get; set; }
//[XmlIgnore]
//public ProjectFile DataFile { get; set; }

[XmlIgnore]
public MainWindowViewModel MVM => DataStorage.MainViewModel;

#endregion

}
}
```

UcViewModelBase.cs

```
using MerlinTest.Tools.TestStudio.Model;
using MerlinTestStudio_Demo_Telerik;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Telerik.Windows.Controls;

namespace MT.TestStudio.GUI.ViewModels
{
/// <summary>
/// Class that represents the base view model for the user controls.
/// </summary>
public abstract class UcViewModelBase : ViewModelBase // : PVM
{
#region Private Member Variables
private bool _saveRequired;
private bool _isDataLoaded;
private string _fileName;
#endregion Private Member Variables
```

```
#region Constructor

public UcViewModelBase()
{
// Set Some Defaults
this.SaveRequired = false;
this.IsDataLoaded = false;
}

#endregion Constructor

#region Public Properties

/// <summary>
/// Gets / Sets if the control requires to be saved.
/// </summary>
public bool SaveRequired
{
get { return _saveRequired; }
set { _saveRequired = value;  OnPropertyChanged("SaveRequired"); }
}

/// <summary>
/// Gets / Sets a variable to indicate if the View-Model has loaded the data stored in the associated
data file.
/// </summary>
public bool IsDataLoaded
{
get { return _isDataLoaded; }
set { _isDataLoaded = value; OnPropertyChanged("IsDataLoaded"); }
}

/// <summary>
/// The Filename and full path of the file accociated with this instance of the viewModel.
/// </summary>
public string FileName
{
get { return _fileName; }
set { _fileName = value; OnPropertyChanged("FileName"); }
}
#endregion Public Properties
```

```
#region Public Methods

/// <summary>
/// Virtual method that when overridden can be used to save data.
/// </summary>
public virtual void SaveDataFile()
{
// Do nothing.
}

/// <summary>
/// Virtual method that when overridden can be used to load data.
/// </summary>
public virtual void LoadDataFile()
{
// Do nothing.
}

#endregion Public Methods

}
}
```

ViewModelBase.cs

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.Base
{
public abstract class ViewModelBase : INotifyPropertyChanged
{
public event PropertyChangedEventHandler PropertyChanged;

protected bool SetProperty<T>(ref T field, T newValue, [CallerMemberName]string propertyName
= null)
{
if (!EqualityComparer<T>.Default.Equals(field, newValue))
{
field = newValue;
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
return true;
```

```
}
return false;
}
}
}
```

AttenuatorResultsViewerVM.cs

```
using MathNet.Numerics;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Input;
using System.Xml.Serialization;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public class AttenuatorResultsViewerVM : PVM
{
private List<KvPAttenuationCalFactorKeyListOfKvPOfDoubleDouble> Data = new
List<KvPAttenuationCalFactorKeyListOfKvPOfDoubleDouble>();
public string resultParentName = string.Empty;
public string resultName = string.Empty;

#region Private members
private double _selectedFrequency;
private string _selectedAmpConfig;
private string _selectedGroup;
private string _selectedWaveform;
private double _selectedLevel;
private List<double> _availableFrequencies = new List<double>();
private List<string> _availableAmpConfigs = new List<string>();
private List<string> _availableGroups = new List<string>();
private List<string> _availableWaveforms = new List<string>();
private List<double> _availableLevels = new List<double>();
private List<KvPOfDoubleDouble> _chartData = new List<KvPOfDoubleDouble>();
private List<double> _regressionCoefficents = new List<double>();
private string _regressionFormula;
#endregion
```

```
#region Public members

#region Chart Data Filters
public double SelectedFrequency
{
get { return _selectedFrequency; }
set
{
_selectedFrequency = value;
OnPropertyChanged("SelectedFrequency");
///Frequncies don't need data filtering due to the fact their already sperated disticntly with data.
///
//Filters the amp configs with Data.
AvailableAmpConfigs = (Data.Where(x => x.Frequency == SelectedFrequency).Select(x =>
x.AmpConfiguration)).Distinct().ToList();
SelectedAmpConfig = null;
//Filters the Groups with Data.
AvailableGroups = (Data.Where(x => x.AmpConfiguration == SelectedAmpConfig && x.Frequency
== SelectedFrequency).Select(x => x.Group)).Distinct().ToList();
SelectedGroup = null;
//Filters the levels with data.
AvailableLevels = (Data.Where(x => x.Group == SelectedGroup &&  x.AmpConfiguration ==
SelectedAmpConfig && x.Frequency == SelectedFrequency ).Select(x =>
x.Level)).Distinct().ToList();
SelectedLevel = 0;
}
}
public string SelectedAmpConfig
{
get { return _selectedAmpConfig; }
set
{
_selectedAmpConfig = value;
OnPropertyChanged("SelectedAmpConfig");

//Filters the Groups with Data.
AvailableGroups = (Data.Where(x => x.AmpConfiguration == SelectedAmpConfig && x.Frequency
== SelectedFrequency).Select(x => x.Group)).Distinct().ToList();
SelectedGroup = null;
//Filters the levels with data.
AvailableLevels = (Data.Where(x => x.Group == SelectedGroup && x.AmpConfiguration ==
SelectedAmpConfig && x.Frequency == SelectedFrequency).Select(x =>
```

```
x.Level)).Distinct().ToList();
SelectedLevel = 0;
}
}
public string SelectedGroup
{
get { return _selectedGroup; }
set
{
_selectedGroup = value;
OnPropertyChanged("SelectedGroup");

//Filters the levels with data.
AvailableLevels = (Data.Where(x => x.Group == SelectedGroup && x.AmpConfiguration ==
SelectedAmpConfig && x.Frequency == SelectedFrequency).Select(x =>
x.Level)).Distinct().ToList();
SelectedLevel = 0;
}
}
public double SelectedLevel
{
get { return _selectedLevel; }
set
{
_selectedLevel = value;
OnPropertyChanged("SelectedLevel");
}
}
public string SelectedWaveform
{
get { return _selectedWaveform; }
set
{
_selectedWaveform = value;
OnPropertyChanged("SelectedWaveform");
}
}
#endregion

#region Available Filters
public List<double> AvailableFrequencies
{
```

```
get { return _availableFrequencies; }
set
{
_availableFrequencies = value;
OnPropertyChanged("AvailableFrequencies");
}
}
public List<string> AvailableAmpConfigs
{
get { return _availableAmpConfigs; }
set
{
_availableAmpConfigs = value;
OnPropertyChanged("AvailableAmpConfigs");
}

}
public List<string> AvailableGroups
{
get { return _availableGroups; }
set
{
_availableGroups = value;
OnPropertyChanged("AvailableAmpConfigs");
}
}
public List<string> AvailableWaveforms
{
get { return _availableWaveforms; }
set
{
_availableWaveforms = value;
OnPropertyChanged("AvailableWaveforms");
}
}
public List<double> AvailableLevels
{
get { return _availableLevels; }
set
{
_availableLevels = value;
OnPropertyChanged("AvailableLevels");
```

```
}
}
#endregion

public List<KvPOfDoubleDouble> ChartData
{
get { return _chartData; }
set { _chartData = value; OnPropertyChanged("ChartData");
OnPropertyChanged("DeltaChartData"); OnPropertyChanged("IsChartDataLoaded"); }
}
public List<KvPOfDoubleDouble> DeltaChartData
{
get
{
var deltaData = new List<KvPOfDoubleDouble>(ChartData);
if (deltaData.Count > 0)
deltaData.RemoveAt(0);
return deltaData;
}
}
public bool IsChartDataLoaded { get { return ChartData != null ? ChartData.Count > 0 : false; } }
public List<double> RegressionCoefficents
{
get { return _regressionCoefficents; }
set { _regressionCoefficents = value; OnPropertyChanged("RegressionCoefficents"); }
}
public string RegressionFormula
{
get { return _regressionFormula; }
set { _regressionFormula = value; OnPropertyChanged("RegressionFormula"); }
}
public bool ShowPolynomialRegressionControls { get { return resultName ==
"srcPortRF110FineGrainAttenuatorResults" ? true : false; } } //Doesn't show for course grain
results.

public string HorizontalAxisTitle { get { return ShowPolynomialRegressionControls ? "Voltage (V)" :
"SrcAttenCalFactor (dB)"; } }
#endregion

private CalResult _setData;
public CalResult ResultDataToExtrapolate
{
```

```
get { return _setData; }
set
{
_setData = value;
OnPropertyChanged("ResultDataToExtrapolate");

//Code from Constructor.
}
}

//Data accessor for Cal Def Info data bindings and Model parent matching.
public CalDataModel CalDataInfo { get; set; }

#region Constructor
public override void PaneClose()
{
//this.View = null; //No View to nullify. Already unloads UC on pane close.
}

/// <summary>
/// Constructor.
/// </summary>
/// <param name="calResult"></param>
public AttenuatorResultsViewerVM(Type contentType, CalResult calResult, CalDataModel
calDataInfo) : base(contentType)
{
resultParentName = calResult.CalDataModelParent;
resultName = calResult.ResultName;
this.CalDataInfo = calDataInfo; //Set parent. used for pane removal.

//Convert cal result data from an interface to the correct type casted object.
Data.AddRange(calResult.CalFactorKeys.Select(cfk => cfk as
KvPAttenuationCalFactorKeyListOfKvPOfDoubleDouble));

#region Distinct Filter Selection
AvailableFrequencies = (Data.Select(x => x.Frequency).Distinct().ToList());
if (AvailableFrequencies.Count() > 0)
SelectedFrequency = AvailableFrequencies[0];

AvailableAmpConfigs = (Data.Select(x => x.AmpConfiguration).Distinct().ToList());
if (AvailableAmpConfigs.Count() > 0)
SelectedAmpConfig = AvailableAmpConfigs[0];
```

```
AvailableGroups = (Data.Select(x => x.Group).Distinct().ToList());
if (AvailableGroups.Count() > 0)
SelectedGroup = AvailableGroups[0];

AvailableWaveforms = (Data.Select(x => x.Waveform).Distinct().ToList());
if (AvailableWaveforms.Count() > 0)
SelectedWaveform = AvailableWaveforms[0];

AvailableLevels = (Data.Select(x => x.Level).Distinct().ToList());
if (AvailableLevels.Count() > 0)
SelectedLevel = AvailableLevels[0];
#endregion

ChangeChartData("Group"); //Initial call to change chart data to the first set of available data.
}

#endregion Constructor

#region Commands
public ICommand ChangeChartDataCommand { get { return new
DelegateCommand(ChangeChartData); } }
public ICommand RegressionChangedCommand { get { return new
DelegateCommand(RegressionChanged); } }

private void ChangeChartData(object obj)
{
//Gets the data prodvided via the selection filters.
var chart_data = new List<KvPOfDoubleDouble>();
foreach (var x in Data.Where(x => x.Frequency == SelectedFrequency && x.AmpConfiguration ==
SelectedAmpConfig && x.Group == SelectedGroup && SelectedWaveform == x.Waveform &&
SelectedLevel == x.Level))
{
chart_data = x.ListOfKvPOfDoubleDouble;
RegressionCoefficents = x.Coefficents;
}

//Set the chart data with reordered kv factors by the representational X (key).
ChartData = new List<KvPOfDoubleDouble>(chart_data.OrderBy(x => x.Key));

RegressionChanged(null);
```

```
#region Previous Code
//if (SelectedFrequency != 0 && !string.IsNullOrEmpty(SelectedAmpConfig) &&
!string.IsNullOrEmpty(SelectedGroup)  && SelectedLevel != 0) //&&
!string.IsNullOrEmpty(SelectedWaveform)
//{
//    var chart_data = new List<KvPOfDoubleDouble>();
//    foreach (var x in Data.Where(x => x.Frequency == SelectedFrequency && x.AmpConfiguration
== SelectedAmpConfig && x.Group == SelectedGroup && SelectedWaveform == x.Waveform &&
SelectedLevel == x.Level))
//    {
//        chart_data = x.ListOfKvPOfDoubleDouble;
//        RegressionCoefficents = x.Coefficents;
//    }
//
//    //Set the chart data with reordered kv factors by the representational X (key).
//    ChartData = new List<KvPOfDoubleDouble>(chart_data.OrderBy(x => x.Key));
//
//    RegressionChanged(null);
//}
#endregion
}

public void RegressionChanged(object obj)
{
double[] ydata = new double[ChartData.Count];
for (int i = 0; i < ChartData.Count; i++) { ydata[i] = ChartData[i].Value; }

double[] fit = RegressionCoefficents.ToArray();

//Recreate the regression equation to solve for X.
string equation = "x =";
for (int i = 0; i < fit.Length; i++)
{
if (i == 0) { equation += string.Format(" {0} +", fit[i]); }
else if (i == fit.Length - 1) { equation += string.Format(" {0}y^{1}", fit[i], i); }
else { equation += string.Format(" {0}y^{1} +", fit[i], i); }
}
RegressionFormula = equation;

//Get regressed X Data.
for (int i = 0; i < ydata.Length; i++) { ChartData[i].Regressed = Polynomial.Evaluate(ydata[i], fit); }
}
```

```
#endregion

}
}
```

CalibrationConfiguratorVM.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Input;
using Telerik.Windows.Controls;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using UnitConversionLib;
using MT.TestStudio.GUI.ViewModels;
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTest.Common.Types;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public interface MainView
{
/// <summary>
/// Clears the focus on the GridView respresenting CalPoint Editor.
/// </summary>
void UnselectCalPoints();

/// <summary>
///Fire when any of the save/export commands are fired, this Forces the grid to commit any
pending edits from cells in edit mode.
/// </summary>
void ForceCommitEdit();

/// <summary>
///
/// </summary>
void GridViewInvalidateMeasure();
}

public class CalibrationConfiguratorVM : PVM
```

```
{
#region Private Members
private string _configStatus = "Create new or open an existing file before editing.";
private CalDataModel _CurrentCalConfig = null;

private CalPointModel _SelectedCalPoint = null;
private ObservableCollection<object> _manySelectedCalPoints = new
ObservableCollection<object>();

#endregion

#region Public Members
private MainView _view;
public MainView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");

if(value != null)
{
CurrentCalConfig.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}
public ObservableCollection<Data.PinModel> AvailablePins { get { return
ParentProject.PinMapDataModel.RfPins; } }

public string ConfigStatus
{
get { return _configStatus; }
set { _configStatus = value; OnPropertyChanged("ConfigStatus"); }
}
public CalDataModel CurrentCalConfig
{
get { return _CurrentCalConfig; }
set
{
if(value != null)
{
```

```
_CurrentCalConfig = value;
OnPropertyChanged("CurrentCalConfig");
}
}
}
public ObservableCollection<object> ManySelectedCalPoints
{
get { return _manySelectedCalPoints; }
}
public CalPointModel SelectedCalPoint
{
get { return _SelectedCalPoint; }
set { _SelectedCalPoint = value; OnPropertyChanged("SelectedCalPoint"); }
}

//Temp
private List<string> _unitList = new List<string>() { "Hz", "KHz", "MHz", "GHz" };
//Temp.
public IEnumerable<string> UnitList { get { return _unitList; } }

public List<KeyValuePair<DoAttenCalEnum, bool>> DoAttenCalKvPs { get; set; }
public List<KeyValuePair<AttenuationSettings, string>> AttenuationSettingsKvPs { get; set; }
public List<KeyValuePair<LfMeasureAmp, string>> LfMeasureAmpKvPs { get; set; }
public List<KeyValuePair<HfAmp, string>> HfAmpKvPs { get; set; }
public List<KeyValuePair<LfAllAmp, string>> LfAllAmpKvPs { get; set; }
public List<KeyValuePair<HfAllAmp, string>> HfAllAmpKvPs { get; set; }
#endregion

#region Constructor
//public event Action CommitEditBeforeSave;
//public event Action ChangeOccured;

public override void PaneClose()
{
this.CurrentCalConfig.CommitEditBeforeSave -= View.ForceCommitEdit;
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
```

```
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
if (this.CurrentCalConfig != null)
{
this.CurrentCalConfig.FilePath = value;
}
}
}

/// <summary>
/// CalibrationConfiguratorVM constructor.
/// </summary>
public CalibrationConfiguratorVM(Type contentType, string dataFilePath, MerlinProject
parentProject, ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject,
parentFolder)
{
//Set the default value....
_CurrentCalConfig = new CalDataModel(parentProject, this.ParentProject.UserTargetSystem)
{
FilePath = dataFilePath,
TestProgram =  parentProject.TestProgram,
Product = parentProject.ProductName
};
//OnChangesMade += (UserControl) =>
//{
//    ///_viewModelLocator.OnControlInteract(UserControl);
//    ///SaveRequired = true;
//
//    IsSaveRequired = true;
//};
////_CurrentCalConfig = new CalDataModel(UserSystemTarget);
////_CurrentCalConfig.Product = this.ProductName; _CurrentCalConfig.TestProgram =
this.TestProgram;
//
//this.CurrentCalConfig = (CalDataModel)dataObject;
//
////Temp. Workaround for future backwards compatibility purposes
//DataFilePath = dataObject.FilePath;
```

```
this.DataFilePath = dataFilePath;

this.LoadData();

this.CurrentCalConfig.ChangeOccured += ChangeOccuredSubscriber;

////////////////////////////////////////////////////////////////////////////////

DoAttenCalKvPs = new List<KeyValuePair<DoAttenCalEnum, bool>>()
{
new KeyValuePair<DoAttenCalEnum, bool>(DoAttenCalEnum.Yes, true),
new KeyValuePair<DoAttenCalEnum, bool>(DoAttenCalEnum.No, false),
//new KeyValuePair<DoAttenCalEnum, bool?>(DoAttenCalEnum.Min_Atten, null)
};

//Make the DoAttenCal KvPs for the SrcCalAttenSettings (Addition to DoAttenCal).
AttenuationSettingsKvPs = new List<KeyValuePair<AttenuationSettings, string>>();
AttenuationSettingsKvPs.AddRange(from AttenuationSettings x in
Enum.GetValues(typeof(AttenuationSettings))
select new KeyValuePair<AttenuationSettings, string>(x, x.ToString()));

//Create the CalPoint nullable enum options without NOP.
LfMeasureAmpKvPs = new List<KeyValuePair<LfMeasureAmp, string>>();
LfMeasureAmpKvPs.AddRange(from LfMeasureAmp x in
Enum.GetValues(typeof(LfMeasureAmp))
where x != LfMeasureAmp.NOP
select new KeyValuePair<LfMeasureAmp, string>(x, x.ToString()));
HfAmpKvPs = new List<KeyValuePair<HfAmp, string>>();
HfAmpKvPs.AddRange(from HfAmp x in Enum.GetValues(typeof(HfAmp))
where x != HfAmp.NOP
select new KeyValuePair<HfAmp, string>(x, x.ToString()));
LfAllAmpKvPs = new List<KeyValuePair<LfAllAmp, string>>();
LfAllAmpKvPs.AddRange(from LfAllAmp x in Enum.GetValues(typeof(LfAllAmp))
where x != LfAllAmp.NOP
select new KeyValuePair<LfAllAmp, string>(x, x.ToString()));
HfAllAmpKvPs = new List<KeyValuePair<HfAllAmp, string>>();
HfAllAmpKvPs.AddRange(from HfAllAmp x in Enum.GetValues(typeof(HfAllAmp))
where x != HfAllAmp.NOP
select new KeyValuePair<HfAllAmp, string>(x, x.ToString()));
}
```

```
public override void SaveData()
{
ForceCommitEdit();
//CommitEditBeforeSave?.Invoke();

this.CurrentCalConfig.SaveToXml(this.DataFilePath);

//this.CompileCalDataV5(); //NEED CONFIRMATION. (COMPILE AFTER SAVE ON RFFE PATTERNS TOO)

Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public void ForceCommitEdit()
{
if(View != null)
{
View.ForceCommitEdit();
}
}

public override void LoadData()
{
CalDataModel dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = CalDataModel.LoadFromXml(this.DataFilePath);
}

if (dataModel != null)
{
this.CurrentCalConfig = dataModel;

//IFileData stuff
dataModel.FilePath = this.DataFilePath;
dataModel.ParentProject = this.ParentProject;
dataModel.TestProgram = this.ParentProject.TestProgram;
dataModel.Product = this.ParentProject.TestProgram;
dataModel.GlobalSystemVarient = this.ParentProject.UserTargetSystem;
```

```
//MUST APPLY TO ALL PVMs USING IFILEDATA OBJECTS.(WILL cause memory issue if
events are not handled)!
//Look for IFileData object in project collection and remove any existing ones with save file path
//Before adding the new one.
#region Duplicate IFileData object removal TEMP BUG FIX
//List<CalDataModel> duplicateModels = new List<CalDataModel>();
//foreach (var cdm in this.ParentProject.CalibrationDefinitions)
//{
//    if(cdm.FilePath == this.DataFilePath)
//    {
//        duplicateModels.Add(cdm);
//    }
//}
//foreach(var cdmToRemove in duplicateModels)
//{
//    this.ParentProject.CalibrationDefinitions.Remove(cdmToRemove);
//}
#endregion Duplicate IFileData object removal TEMP BUG FIX

this.ParentProject.CalibrationDefinitions.Add(dataModel);
}

base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{
foreach (CalPointModel cpm in this.CurrentCalConfig.CalPointModelCollection)
{
//If SrcPinAlias is null, find pin via PinName, otherwise find it via PinAlias.
int srcPinIndex = this.ParentProject.PinMapDataModel.RfPins.ToList().FindIndex(pin =>
string.IsNullOrEmpty(cpm.SrcPinAlias) ? pin.PinName == cpm.SrcPinName : pin.PinAlias ==
cpm.SrcPinAlias);
if (srcPinIndex != -1)
{
cpm.SrcPin = this.ParentProject.PinMapDataModel.RfPins[srcPinIndex];
}

//If MeasPinAlias is null, find pin via PinName, otherwise find it via PinAlias.
int measPinIndex = this.ParentProject.PinMapDataModel.RfPins.ToList().FindIndex(pin =>
```

```
string.IsNullOrEmpty(cpm.MeasPinAlias) ? pin.PinName == cpm.MeasPinName : pin.PinAlias ==
cpm.MeasPinAlias);
if (measPinIndex != -1)
{
cpm.MeasPin = this.ParentProject.PinMapDataModel.RfPins[measPinIndex];
}

#region Previous Code before Pin Alias was needed
//foreach (Data.PinModel pin in projectPinMap.RfPins.Where(pin => pin.PinName ==
cpm.SrcPinName))
//{
//    cpm.SrcPin = pin;
//}
//foreach (Data.PinModel pin in projectPinMap.RfPins.Where(pin => pin.PinName ==
cpm.MeasPinName))
//{
//    cpm.MeasPin = pin;
//}
#endregion Previous Code before Pin Alias was needed
}
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
//public void OnChangeOccured()
//{
//    ChangeOccured?.Invoke();
//}

public override void Undo()
{
if (this.CurrentCalConfig != null)
{
this.CurrentCalConfig.Undo();
}

base.Undo();
}
public override void Redo()
{
if(this.CurrentCalConfig != null)
{
this.CurrentCalConfig.Redo();
```

```
}

base.Redo();
}

#endregion

#region Commands

#region User Cal Preview
public ICommand UserCalPreviewCommand { get { return new RelayCommand(UserCalPreview);
} }
private void UserCalPreview()
{
View.ForceCommitEdit();

try
{
//Check
if (ValidateCalPointPinsData() == false)
{
return;
}
#region Compile actions
CalDataV5 dataOut = CurrentCalConfig.GetProcessedDataObject();

List<Data.MappingModel> firstSiteMappings = new List<Data.MappingModel>(); //List of 1st site
pin mappings.
foreach (Data.PinModel pin in ParentProject.PinMapDataModel.RfPins)
{
var PinMapOne = pin.mappings.First();
if (PinMapOne != null)
{
firstSiteMappings.Add(PinMapOne);
}
}
if (firstSiteMappings.Count() == ParentProject.PinMapDataModel.RfPins.Count()) //Ensure no
mappings are missing or null
{
List<CalPointV5> ProcessedCalPoints = new List<CalPointV5>();

//Single Site CalPointV5 Generation
```

```
int pointID = 0;
foreach (CalPointModel cpmBase in CurrentCalConfig.CalPointModelCollection)
{
CalPointV5 GenCalPoint = cpmBase.CreateCalPointV5();
int srcPinIndex = cpmBase.SrcPin != null ?
ParentProject.PinMapDataModel.RfPins.IndexOf(cpmBase.SrcPin) : 0; //Should never be zero.
int measPinIndex = cpmBase.MeasPin != null ?
ParentProject.PinMapDataModel.RfPins.IndexOf(cpmBase.MeasPin) : 0; //Should never be zero.
string srcPortFound = firstSiteMappings[srcPinIndex].IO;
string measPortFound = firstSiteMappings[measPinIndex].IO;
GenCalPoint.ID = pointID;
GenCalPoint.IsVnaExtra = false;
GenCalPoint.SourcePort = (SrcPort)Enum.Parse(typeof(SrcPort), srcPortFound, true);
GenCalPoint.MeasurePort = (MeasPort)Enum.Parse(typeof(MeasPort), measPortFound, true);
ProcessedCalPoints.Add(GenCalPoint);

pointID++;
}
//VNA Addition Extrapolation
List<CalPointV5> vnaCalPoints = (ProcessedCalPoints.Where(cp5 => cp5.CalibrationType ==
CalType.VNA_1P || cp5.CalibrationType == CalType.VNA_2P)).ToList();
List<double> uniqueVnaFrequencies = vnaCalPoints.GroupBy(x => x.Frequency).Select(group =>
group.Key).ToList();
foreach (CalPointV5 vnacp in vnaCalPoints)
{
CalType vnatype = vnacp.CalibrationType;
string srcPinName = vnacp.SrcPinName;
string measPinName = vnacp.MeasPinName;

List<CalPointV5> filteredVnaCalPoints = ProcessedCalPoints.Where(x => x.CalibrationType ==
vnatype && x.SrcPinName == srcPinName && x.MeasPinName == measPinName).ToList();

foreach (double freq in uniqueVnaFrequencies)
{
CalPointV5 foundFreqCP = filteredVnaCalPoints.Find(i => i.Frequency == freq);
if (foundFreqCP == null) //CalPoint with one of the frequencies is does not exist, need to generate
and place correctly.
{
int insertIndex = ProcessedCalPoints.IndexOf(vnacp) + 1;
CalPointV5 vnacpClone = CalPointModel.CloneCalPointV5(vnacp);
vnacpClone.Frequency = freq; //Change the cloned calpoint's frequency before main data
insertion.
```

```
vnacpClone.IsVnaExtra = true; //Mark as generated item to avoid confusion any on re-import(s).
ProcessedCalPoints.Insert(insertIndex, vnacpClone);
}
}
}
//Multi Site CalPoint Generation (Final Step).
foreach (CalPointV5 cp5 in ProcessedCalPoints)
{
int numOfSites = ParentProject.PinMapDataModel.RfPins.Max((x) => x.mappings.Count);
for (int site = 0; site < numOfSites; site++)
{
//string realSrcPinName = string.IsNullOrEmpty(cp5.SrcPinNameAlias) ? cp5.SrcPinName :
cp5.SrcPinNameAlias;
Data.PinModel srcPinFind = ParentProject.PinMapDataModel.RfPins.ToList()
.Find(i => (string.IsNullOrEmpty(i.PinAlias) ? i.PinName : i.PinAlias) == cp5.SrcPinName);

//string realMeasPinName = string.IsNullOrEmpty(cp5.MeasPinNameAlias) ? cp5.MeasPinName :
cp5.MeasPinNameAlias;
Data.PinModel measPinFind = ParentProject.PinMapDataModel.RfPins.ToList()
.Find(i => (string.IsNullOrEmpty(i.PinAlias) ? i.PinName : i.PinAlias) == cp5.MeasPinName);

string srcPortFound = srcPinFind.mappings[site].IO;
string measPortFound = measPinFind.mappings[site].IO;

CalPointV5 SiteCalPoint = CalPointModel.CloneCalPointV5(cp5);
SiteCalPoint.SiteNumber = site + 1;
SiteCalPoint.SourcePort = (SrcPort)Enum.Parse(typeof(SrcPort), srcPortFound, true);
SiteCalPoint.MeasurePort = (MeasPort)Enum.Parse(typeof(MeasPort), measPortFound, true);

dataOut.CalPoints.Add(SiteCalPoint);
}
}
}
else
{
MessageBox.Show("Missing or Null 1st site mapping");
}
#endregion
RadWindow exportView = new RadWindow()
{
Header = "CalDataV5 UserCal Preview",
Height = 800,
```

```
Width = 1000,
WindowStartupLocation = WindowStartupLocation.CenterScreen,
Content = new UserControls.CalDataInfo() { DataContext = dataOut }
};
exportView.Show();

}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
#endregion

//Old
#region OLD Import CalDataV5 (Unused)
public ICommand ImportCalDataV5Command { get { return new
DelegateCommand(ImportCalDataV5); } }
private void ImportCalDataV5(object obj)
{
//string ExtReq = obj as string;
//switch (ExtReq)
//{
//    case ".xml":
//        Microsoft.Win32.OpenFileDialog xml_dlg = new Microsoft.Win32.OpenFileDialog()
//        {
//            DefaultExt = ".xml",
//            Filter = "XML File (.xml)|*.xml",
//            Title = "Import CalDataV5",
//        };
//        if (xml_dlg.ShowDialog() == true)
//        {
//            CalDataV5 CalDataObj =
CalDataV5.ImportCalDataXMLNoModifyCheck(xml_dlg.FileName);
//
//            if (CalDataObj.ResultsData == true) //Results file, don't allow import.
//            {
//                MessageBox.Show("Calibration Definiton file invalid!(Calibration Results File format)");
//            }
//            else //Config file, allow import.
//            {
//                //List<string> nonExistentPins = new List<string>();
//                //foreach (CalPointV5 cpBase in CalDataObj.CalPoints) //Search for missing pins.
//                //{
//                //    if (PVM.ParentProject.Pins.ToList().Find(x => x.PinName == cpBase.SrcPinName)
```

```
== null)
//              //    { nonExistentPins.Add(cpBase.SrcPinName); }
//              //    if (PVM.ParentProject.Pins.ToList().Find(x => x.PinName ==
cpBase.MeasPinName) == null)
//              //    { nonExistentPins.Add(cpBase.MeasPinName); }
//              //}
//              //List<string> duplicateMissingPins = nonExistentPins.GroupBy(x => x).Where(group
=> group.Count() > 1).Select(group => group.Key).ToList();
//              //if (duplicateMissingPins.Count() != 0)
//              //{
//              //    string missingPinsMsg = "The following pins are missing from the current PinMap:
";
//              //    foreach(string mps in duplicateMissingPins)
//              //    {
//              //        missingPinsMsg += string.Format("\n{0}", mps);
//              //    }
//              //    MessageBox.Show(missingPinsMsg);
//              //    //Need to edit the usercontrol and figure out the continuation of the import logic...
//              //    ///RadWindow exportView = new RadWindow()
//              //    ///{
//              //    ///    Header = "Would you like to auto-generate missing pins?",
//              //    ///    Height = 800,
//              //    ///    Width = 800,
//              //    ///    WindowStartupLocation = WindowStartupLocation.CenterScreen,
//              //    ///    Content = new UserControls.Tools.GeneratePinsFromCalibrationDefinition() {
DataContext = this }
//              //    ///};
//              //    ///exportView.Show();
//              //}
//
//              ObservableCollection<CalPointModel> CalPointModelData = new
ObservableCollection<CalPointModel>();
//              if (CalDataObj.CalibrationDataVersion >= 6) //Includes site numbers
//              {
//                  foreach (CalPointV5 cpBase in CalDataObj.CalPoints)
//                  {
//                      if (cpBase.IsVnaExtra == false && cpBase.SiteNumber == 1) //Requested that
VnaExtras not be imported.
//                      {
//
//                          Data.Pin srcPin = PVM.ParentProject.PinMapDataModel.RfPins.ToList().Find(x
=> x.mappings.First().IO == cpBase.SourcePort.ToString());
```

```
//                      Data.Pin measPin =
PVM.ParentProject.PinMapDataModel.RfPins.ToList().Find(x => x.mappings.First().IO ==
cpBase.MeasurePort.ToString());
//
//                      CalPointModel newCpm = new CalPointModel(cpBase,
CurrentCalConfig.GlobalSystemVarient, "Hz");
//
//                      int numOfSites = PVM.ParentProject.PinMapDataModel.RfPins.Count() > 0 ?
PVM.ParentProject.PinMapDataModel.RfPins.Max((x) => x.mappings.Count) : 0;
//                      if (numOfSites != 0)
//                      {
//                          newCpm.SrcPin = srcPin;
//                          newCpm.MeasPin = measPin;
//
//                          ///Check if pin-ports match for already existing name-matched pins ?
//
//                          CalPointModelData.Add(newCpm);
//                      }
//                      else //Create the CalPointModel with all other information except Src/Meas
pin(s)
//                      {
//                          CalPointModelData.Add(newCpm);
//                      }
//                  }
//              }
//              else //only includes VNA extras, no multi site generation importing
//              {
//                  foreach (CalPointV5 cpBase in CalDataObj.CalPoints)
//                  {
//                      if (cpBase.IsVnaExtra == false) //Requested that VnaExtras not be imported.
//                      {
//                          Data.Pin srcPin = PVM.ParentProject.PinMapDataModel.RfPins.ToList().Find(x
=> x.mappings.First().IO == cpBase.SourcePort.ToString());
//                          Data.Pin measPin =
PVM.ParentProject.PinMapDataModel.RfPins.ToList().Find(x => x.mappings.First().IO ==
cpBase.MeasurePort.ToString());
//
//                          CalPointModel newCpm = new CalPointModel(cpBase,
CurrentCalConfig.GlobalSystemVarient, "Hz");
//
//                          int numOfSites = PVM.ParentProject.PinMapDataModel.RfPins.Count() > 0 ?
```

```
PVM.ParentProject.PinMapDataModel.RfPins.Max((x) => x.mappings.Count) : 0;
//                      if (numOfSites != 0)
//                      {
//                          newCpm.SrcPin = srcPin;
//                          newCpm.MeasPin = measPin;
//
//                          ///Check if pin-ports match for already existing name-matched pins ?
//
//                          CalPointModelData.Add(newCpm);
//                      }
//                      else //Create the CalPointModel with all other information except Src/Meas
pin(s)
//                      {
//                          CalPointModelData.Add(newCpm);
//                      }
//                  }
//              }
//          }
//
//          //Set the imported data.
//          CurrentCalConfig = new CalDataModel(PVM.ParentProject, CalDataObj, FileName,
UserSystemTarget);
//          CurrentCalConfig.Product = this.ProductName; CurrentCalConfig.TestProgram =
this.TestProgram;
//          foreach (var icpm in CalPointModelData) {
CurrentCalConfig.CalPointModelCollection.Add(icpm); } //Don't replace collection, Add imported
calpoints to maintain reference to the HistoryManager.
//          ForceChangesMade();
//      }
//  }
//  break;
//case ".csv":
//  MessageBox.Show("Warning: Pin Map (1st site only) must be built before importing, \n
otherwise non-mapped (Src & Meas port) CalPoint(s) will not be imported.");
//  Microsoft.Win32.OpenFileDialog csv_dlg = new Microsoft.Win32.OpenFileDialog()
//  {
//      DefaultExt = ".csv",
//      Filter = "CSV File (.csv)|*.csv",
//      Title = "Import Legacy CalDataV5",
//  };
//  if (csv_dlg.ShowDialog() == true)
//  {
```

```
//          CalDataV5 CalDataObj = CalDataV5.ImportCalDataCSV(csv_dlg.FileName);
//          ObservableCollection<CalPointModel> CalPointModelData = new
ObservableCollection<CalPointModel>();
//
//          foreach (CalPointV5 cpBase in CalDataObj.CalPoints)
//          {
//              Data.Pin srcPin = PVM.ParentProject.PinMapDataModel.RfPins.ToList().Find(x =>
x.mappings.First().IO == cpBase.SourcePort.ToString());
//              Data.Pin measPin = PVM.ParentProject.PinMapDataModel.RfPins.ToList().Find(x =>
x.mappings.First().IO == cpBase.MeasurePort.ToString());
//
//              if (srcPin != null || measPin != null)
//              {
//                  CalPointModel newCpm = new CalPointModel(cpBase,
CurrentCalConfig.GlobalSystemVarient, "Hz");
//
//                  int numOfSites = PVM.ParentProject.PinMapDataModel.RfPins.Count() > 0 ?
PVM.ParentProject.PinMapDataModel.RfPins.Max((x) => x.mappings.Count) : 0;
//                  if (numOfSites != 0)
//                  {
//                      newCpm.SrcPin = srcPin;
//                      newCpm.MeasPin = measPin;
//
//                      ///Check if pin-ports match for already existing name-matched pins ?
//
//                      CalPointModelData.Add(newCpm);
//                  }
//                  else //Create the CalPointModel with all other information except Src/Meas pin(s)
//                  {
//                      CalPointModelData.Add(newCpm);
//                  }
//              }
//          }
//
//          //Set the imported data.
//          CurrentCalConfig = new CalDataModel(PVM.ParentProject, CalDataObj, FileName,
UserSystemTarget);
//          CurrentCalConfig.Product = this.ProductName; CurrentCalConfig.TestProgram =
this.TestProgram;
//          foreach (var icpm in CalPointModelData) {
CurrentCalConfig.CalPointModelCollection.Add(icpm); } //Don't replace collection, Add imported
calpoints to maintain reference to the HistoryManager.
```

```
//            ForceChangesMade();
//        }
//        break;
//}
}
#endregion

#region Compile to CalDataV5
public ICommand CompileCalDataV5Command { get { return new
RelayCommand(CompileCalDataV5); } }
private void CompileCalDataV5()
{
try
{
#region Pre-Compile Checks
//Pre-Export Data Checks
if (ValidateCalPointPinsData() == false)
{
MessageBox.Show("Pin Map data not valid, view output for more information...");
return;
}
//Ensure that at least one of the options is checked (only needed on compile, not preview)
if (CurrentCalConfig.CalibrateRF == false && CurrentCalConfig.CalibrateNoise == false &&
CurrentCalConfig.CalibrateVNA == false)
{
MessageBox.Show("Please check at least one of the calibration options:\nCalibrateRF,\nCalibrate
Noise,\nCalibrateVNA");
return;
}
//Ensure that at least one of the Amplifiers to Calibrate is checked.
if (CurrentCalConfig.AmpConfigModelList.Any(obj => obj.IsAdded == true) == false)
{
MessageBox.Show("Please check at least one of the Amplifiers to Calibrate before compiling.");
return;
}
#endregion

#region Compile actions
CalDataV5 dataOut = CurrentCalConfig.GetProcessedDataObject();

List<Data.MappingModel> firstSiteMappings = new List<Data.MappingModel>(); //List of 1st site
pin mappings.
```

```
foreach (Data.PinModel pin in ParentProject.PinMapDataModel.RfPins)
{
var PinMapOne = pin.mappings.First();
if (PinMapOne != null)
{
firstSiteMappings.Add(PinMapOne);
}
}
if (firstSiteMappings.Count() == ParentProject.PinMapDataModel.RfPins.Count()) //Ensure no
mappings are missing or null
{
List<CalPointV5> ProcessedCalPoints = new List<CalPointV5>();

//Single Site CalPointV5 Generation
int pointID = 0;
foreach (CalPointModel cpmBase in CurrentCalConfig.CalPointModelCollection)
{
CalPointV5 GenCalPoint = cpmBase.CreateCalPointV5();
int srcPinIndex = cpmBase.SrcPin != null ?
ParentProject.PinMapDataModel.RfPins.IndexOf(cpmBase.SrcPin) : 0; //Should never be zero.
int measPinIndex = cpmBase.MeasPin != null ?
ParentProject.PinMapDataModel.RfPins.IndexOf(cpmBase.MeasPin) : 0; //Should never be zero.
string srcPortFound = firstSiteMappings[srcPinIndex].IO;
string measPortFound = firstSiteMappings[measPinIndex].IO;
GenCalPoint.ID = pointID;
GenCalPoint.IsVnaExtra = false;
GenCalPoint.SourcePort = (SrcPort)Enum.Parse(typeof(SrcPort), srcPortFound, true);
GenCalPoint.MeasurePort = (MeasPort)Enum.Parse(typeof(MeasPort), measPortFound, true);
ProcessedCalPoints.Add(GenCalPoint);

pointID++;
}
//VNA Addition Extrapolation
List<CalPointV5> vnaCalPoints = (ProcessedCalPoints.Where(cp5 => cp5.CalibrationType ==
CalType.VNA_1P || cp5.CalibrationType == CalType.VNA_2P)).ToList();
List<double> uniqueVnaFrequencies = vnaCalPoints.GroupBy(x => x.Frequency).Select(group =>
group.Key).ToList();
foreach (CalPointV5 vnacp in vnaCalPoints)
{
CalType vnatype = vnacp.CalibrationType;
string srcPinName = vnacp.SrcPinName;
string measPinName = vnacp.MeasPinName;
```

```
List<CalPointV5> filteredVnaCalPoints = ProcessedCalPoints.Where(x => x.CalibrationType ==
vnatype && x.SrcPinName == srcPinName && x.MeasPinName == measPinName).ToList();

foreach (double freq in uniqueVnaFrequencies)
{
CalPointV5 foundFreqCP = filteredVnaCalPoints.Find(i => i.Frequency == freq);
if (foundFreqCP == null) //CalPoint with one of the frequencies is does not exist, need to generate
and place correctly.
{
int insertIndex = ProcessedCalPoints.IndexOf(vnacp) + 1;
CalPointV5 vnacpClone = CalPointModel.CloneCalPointV5(vnacp);
vnacpClone.Frequency = freq; //Change the cloned calpoint's frequency before main data
insertion.
vnacpClone.IsVnaExtra = true; //Mark as generated item to avoid confusion any on re-import(s).
ProcessedCalPoints.Insert(insertIndex, vnacpClone);
}
}
}
//Multi Site CalPoint Generation (Final Step).
foreach (CalPointV5 cp5 in ProcessedCalPoints)
{
int numOfSites = ParentProject.PinMapDataModel.RfPins.Max((x) => x.mappings.Count);
for (int site = 0; site < numOfSites; site++)
{
//string realSrcPinName = string.IsNullOrEmpty(cp5.SrcPinNameAlias) ? cp5.SrcPinName :
cp5.SrcPinNameAlias;
Data.PinModel srcPinFind = ParentProject.PinMapDataModel.RfPins.ToList()
.Find(i => (string.IsNullOrEmpty(i.PinAlias) ? i.PinName : i.PinAlias) == cp5.SrcPinName);

//string realMeasPinName = string.IsNullOrEmpty(cp5.MeasPinNameAlias) ? cp5.MeasPinName :
cp5.MeasPinNameAlias;
Data.PinModel measPinFind = ParentProject.PinMapDataModel.RfPins.ToList()
.Find(i => (string.IsNullOrEmpty(i.PinAlias) ? i.PinName : i.PinAlias) == cp5.MeasPinName);

string srcPortFound = srcPinFind.mappings[site].IO;
string measPortFound = measPinFind.mappings[site].IO;

CalPointV5 SiteCalPoint = CalPointModel.CloneCalPointV5(cp5);
SiteCalPoint.SiteNumber = site + 1;
SiteCalPoint.SourcePort = (SrcPort)Enum.Parse(typeof(SrcPort), srcPortFound, true);
SiteCalPoint.MeasurePort = (MeasPort)Enum.Parse(typeof(MeasPort), measPortFound, true);
```

```
dataOut.CalPoints.Add(SiteCalPoint);
}
}
}
else
{
MessageBox.Show("Missing or Null 1st site mapping");
}
#endregion

#region Overwrite check
string compiledPath =
Path.Combine(this.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Calibration_D
efinitions), $"{Path.GetFileNameWithoutExtension(this.DataFilePath)}.xml");
MessageBoxResult overwrite_result = MessageBoxResult.None;
if (File.Exists(compiledPath))
{
overwrite_result = MessageBox.Show($"{Path.GetFileName(compiledPath)} already exists, Are
you sure you want to overwrite this file?", "Alert", MessageBoxButton.YesNo);
switch (overwrite_result)
{
case MessageBoxResult.Yes:
//Overwrite.
dataOut.SaveToXml(compiledPath);
Console.WriteLine(string.Format("Compiled: {0}", compiledPath));
break;
case MessageBoxResult.No:
//Do nothing with the compiled data above.
return;
}
}
else
{
dataOut.SaveToXml(compiledPath);
Console.WriteLine(string.Format("Compiled: {0}", compiledPath));
}
#endregion Overwrite check
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
```

```
Console.WriteLine(ex.Message);
}

}
#endregion

#region Clear All Cal Points
/// <summary>
/// Clears all the CalPoints from the editor, this action is revertible.
/// </summary>
public ICommand ClearAllCommand { get { return new DelegateCommand(ClearAll); } }
private void ClearAll(object obj)
{
MessageBoxResult messageBoxResult = System.Windows.MessageBox.Show("All unsaved
progress will be lost. Are you sure?", "Clear CalPoints",
System.Windows.MessageBoxButton.YesNo);
if (messageBoxResult == MessageBoxResult.Yes)
{
CurrentCalConfig.CalPointModelCollection.Clear();

this.CurrentCalConfig.OnChangeOccured();
}
}
#endregion

#region Insert Cal Points
/// <summary>
/// Adds a new CalPoint to the editor and logs it (Single-Edit).
/// </summary>
public ICommand InsertCalPointCommand { get { return new DelegateCommand(InsertCalPoint); }
}
private void InsertCalPoint(object obj)
{
List<KeyValuePair<int, object>> itemsInserted = new List<KeyValuePair<int, object>>();
if (obj is ObservableCollection<object>) //Sent by context menu item.
{
ObservableCollection<object> selecteditems = obj as ObservableCollection<object>;
if (selecteditems.Count() <= 0) //No Items selected or source data is empty, insert 1.
{
int insertIndex = 0;
CalPointModel newPoint = new CalPointModel(CurrentCalConfig.GlobalUnit,
CurrentCalConfig.GlobalSystemVarient);
```

```
CurrentCalConfig.CalPointModelCollection.Insert(insertIndex, newPoint);
itemsInserted.Add(new KeyValuePair<int, object>(insertIndex, newPoint));
}
else //Many Items selected, insert the required amount.
{
foreach (var item in selecteditems)
{
int insertIndex = CurrentCalConfig.CalPointModelCollection.IndexOf(item as CalPointModel) + 1;
CalPointModel newPoint = new CalPointModel(CurrentCalConfig.GlobalUnit,
CurrentCalConfig.GlobalSystemVarient);

CurrentCalConfig.CalPointModelCollection.Insert(insertIndex, newPoint);
itemsInserted.Add(new KeyValuePair<int, object>(insertIndex, newPoint));
}
}
CurrentCalConfig.LogInsertAction(itemsInserted);
}

this.CurrentCalConfig.OnChangeOccured();
}
#endregion

#region Remove Cal Points
/// <summary>
/// Removes the selected CalPoint(s) from the editor and logs it (Multi-Edit).
/// </summary>
public ICommand RemoveCalPointsCommand { get { return new
DelegateCommand(RemoveCalPoint); } }
private void RemoveCalPoint(object obj)
{
if(obj is ObservableCollection<object>)
{
ObservableCollection<object> selecteditems = obj as ObservableCollection<object>;
if (selecteditems.Count() == 0) { return; }

ObservableCollection<CalPointModel> itemsToRemove = new
ObservableCollection<CalPointModel>();
List<KeyValuePair<int, object>> items = new List<KeyValuePair<int, object>>();

foreach (var item in selecteditems) //Remove the items from the RadGridView
{
```

```
items.Add(new KeyValuePair<int,
object>(CurrentCalConfig.CalPointModelCollection.IndexOf(item as CalPointModel), item));
itemsToRemove.Add(item as CalPointModel);
}
CurrentCalConfig.LogDeleteAction(items); //Get log data before anything is deleted.
foreach (var item in itemsToRemove)
{
CurrentCalConfig.CalPointModelCollection.Remove(item as CalPointModel);
CurrentCalConfig.PointStatusChanged(item as CalPointModel); //Temp handles errors accordingly
if it's not contained in the collection.
}
}

View.GridViewInvalidateMeasure();

this.CurrentCalConfig.OnChangeOccured();
}
#endregion

#region Duplicate Cal Points
/// <summary>
/// Duplicates the selected CalPoint(s) in the editor and logs it (Multi-Edit).
/// Outputs the duplicate CalPoint underneath the original CalPoint.
/// </summary>
public ICommand DuplicateCalPointCommand { get { return new
DelegateCommand(DuplicateCalPoint); } }
private void DuplicateCalPoint(object obj)
{
if (obj is ObservableCollection<object>)
{
ObservableCollection<object> selecteditems = obj as ObservableCollection<object>;
if (selecteditems.Count() == 0) { return; }
//The line below gets confusing for the user when duplicating multiple items, this edits all of them
at the same time.
//View.ForceCommitEdit(); //Commit any pending edits

List<KeyValuePair<int, object>> items = new List<KeyValuePair<int, object>>();
foreach (var item in selecteditems)
{
int insertIndex = CurrentCalConfig.CalPointModelCollection.IndexOf(item as CalPointModel) + 1;
CalPointModel duplicate = (item as CalPointModel).GetClone();
CurrentCalConfig.CalPointModelCollection.Insert(insertIndex, duplicate);
```

```
items.Add(new KeyValuePair<int, object>(insertIndex, duplicate));
}

CurrentCalConfig.LogDuplicateAction(items);
}

this.CurrentCalConfig.OnChangeOccured();
}
#endregion

#region Unselect Cal Points
//Unselect CalPoints Command
public ICommand ClearCalPointSelectionCommand { get { return new
DelegateCommand(ClearCalPointSelection); } }
private void ClearCalPointSelection(object obj)
{
View.UnselectCalPoints();
}
#endregion

#endregion

#region Methods

private bool ValidateCalPointPinsData()
{
if (CurrentCalConfig.CalPointModelCollection.Count() == 0 ||
ParentProject.PinMapDataModel.RfPins.Count() == 0)
{
MessageBox.Show("CalPoints or Pin Map cannot be empty.");
return false;
}

foreach (CalPointModel cpm in CurrentCalConfig.CalPointModelCollection)
{
//NEEDS to check if the Pin is available in the Pin Map!!! Not all values are null, just unavailable in
the pin map.
if (cpm.SrcPin == null || cpm.MeasPin == null)
{
MessageBox.Show("Missing Src/Meas Pin(s) detected.");
return false;
```

```
//MessageBox.Show("Unable to preview, please fill in the missing SrcPin or MeasPin.");
}
}

//Utilize PinMapData checks
bool isPinMapValid;
ParentProject.PinMapDataModel.ValidatePinMapData(new Enum[] { PinMapSection.Rf_Pins }, out isPinMapValid);
if (isPinMapValid == false)
{
return false;
}

//If all checks pass return true.
return true;
}

//Old
#region Legacy methods
/// <summary>
/// Exports a calibration file in the legacy CSV format using the CalDataModel as the Data Object.
/// </summary>
/// <param name="CalibrationDefinitionFile">The filename and path of the CSV based calibration file.</param>
/// <returns>A calibration object.</returns>
public void ExportCalDataCSV(string filePath)
{
try
{
using (TextWriter csv = new StreamWriter(filePath))
{
csv.WriteLine("{0},{1}", "Product:", CurrentCalConfig.Product);
csv.WriteLine("{0},{1}", "Revision:", CurrentCalConfig.Revision);
csv.WriteLine("{0},{1}", "Test Program:", CurrentCalConfig.TestProgram);
csv.WriteLine("{0},{1}", "Calibration Version:", $"v{CurrentCalConfig.CalibrationDataVersion}");
csv.WriteLine("{0}", "");
csv.WriteLine("{0},{1}", "Calibrate RF:", CurrentCalConfig.CalibrateRF);
csv.WriteLine("{0},{1}", "Calibrate Internal Direct Path:", "");
//CurrentCalConfig.CalibrateInternalDirectPath
csv.WriteLine("{0},{1}", "Calibrate Noise Source:", CurrentCalConfig.CalibrateNoise);
csv.WriteLine("{0},{1}", "Calibrate VNA:", CurrentCalConfig.CalibrateVNA);
csv.WriteLine("{0}", "");
```

```
csv.WriteLine("{0},{1}", "Noise Bandwidth (Hz):", CurrentCalConfig.NoiseCalibrationSampleRate);
csv.WriteLine("{0}", "");
foreach (AmpConfigModel acm in CurrentCalConfig.AmpConfigModelList)
{
csv.WriteLine("{0}, {1}", acm.Config, acm.IsAdded);
}
csv.WriteLine("{0}", "");
csv.WriteLine("{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},{12}", "Source", "Source Port", "Source Port Alias", "Measure Port", "Measure Port Alias", "Use Hf Path", "Frequency", "DeviceExpectedInputLevel", "Do Atten Cal", "Filter", "Meas Atten", "Modulation File", "Comment");
//foreach (CalPointModel c in CurrentCalConfig.CalPointModelCollection)
//{
//    //CalPointModel.ValueUnitModifier(c.Frequency, c.Unit, "Hz")  --- Only needed when using CurrentCalConfig.CalPointModelCollection, otherwise use c.Frequency.
//    csv.WriteLine("{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},{12}", c.Source, c.SourcePort, "", c.MeasurePort, "", ConvertBooleanToYesNo(c.UseHfPath), CalPointModel.ValueUnitModifier(c.Frequency, c.Unit, "Hz"), c.DeviceExpectedInputLevel, ConvertBooleanToYesNo(c.DoAttenCal), ConvertMeasureFilter(c.MeasureFilter), c.MeasureAttenuation, c.Waveform, c.Comment);
//}
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

/// <summary>
/// Converter for legacy true/false expression in CalDataV5.csv
/// </summary>
/// <param name="value"></param>
/// <returns></returns>
private string ConvertBooleanToYesNo(bool value)
{
switch (value)
{
case true: return "Y";
case false: return "N";
default: return "null"; //Signals there has been an Error.
}
}

/// <summary>
/// MeasureFilter value Converter for CalPointModel.
```

```
/// </summary>
/// <param name="mfo"></param>
/// <returns></returns>
private string ConvertMeasureFilter(MeasureFilterOptions mfo)
{
switch (mfo)
{
case MeasureFilterOptions.Bypass: return "BYPASS";
case MeasureFilterOptions.Atten: return "ATTEN";
case MeasureFilterOptions.FILT1: return "HIGH";
case MeasureFilterOptions.FILT3: return "MID";
case MeasureFilterOptions.FILT2: return "LOW";
default: return "null"; //Signals there has been an Error.
}
}

/// <summary>
/// MeasureFilter value Converter for CalPointV5
/// </summary>
/// <param name="mfo"></param>
/// <returns></returns>
private string ConvertMeasureFilter(MeasFilt mfo)
{
switch (mfo)
{
case MeasFilt.Bypass: return "BYPASS";
case MeasFilt.Atten: return "ATTEN";
case MeasFilt.FILT1: return "HIGH";
case MeasFilt.FILT3: return "MID";
case MeasFilt.FILT2: return "LOW";
default: return "null"; //Signals there has been an Error.
}
}
#endregion Legacy methods

#endregion

}

public class AutoPin
{
public bool Generate { get; set; }
```

```
public string AquiredPinName { get; set; }
}
}
```

LimitUcVM.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.CalModels;
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Dynamic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.CalibrationViewModels
{
public interface ILimitUcView
{
void ForceCommitEdit();

void RebindGridView();
}

public class LimitUcVM : PVM
{

//Temp
private List<string> _unitList = new List<string>() { "Hz", "KHz", "MHz", "GHz" };
//Temp.
public IEnumerable<string> UnitList { get { return _unitList; } }

private string _viewingUnit = "GHz";
public string ViewingUnit
{
get { return _viewingUnit; }
set { _viewingUnit = value; OnPropertyChanged("ViewingUnit"); }
```

```
}

private ILimitUcView _view;
public ILimitUcView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");

if (value != null)
{
CalLimits.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}

private CalLimitsModel _dataObject = new CalLimitsModel();
public CalLimitsModel CalLimits
{
get { return _dataObject; }
set
{
_dataObject = value; OnPropertyChanged("CalLimits");
if(_dataObject != null && _dataObject.Results != null)
{
SelectedCalResult = (CalResult)_dataObject.Results.FirstOrDefault(); //Set the first cal result to
display.
}
}
}

private CalResult _selectedCalResult;
public CalResult SelectedCalResult
{
get { return _selectedCalResult; }
set { _selectedCalResult = value; OnPropertyChanged("SelectedCalResult"); }
}

private double _limitMin = CalLimitsModel.DefaultMinTolerance;
public double LimitMin
```

```
{
get { return _limitMin;  }
set { _limitMin = value; OnPropertyChanged("LimitMin"); }
}

private double _limitMax = CalLimitsModel.DefaultMaxTolerance;
public double LimitMax
{
get { return _limitMax; }
set { _limitMax = value; OnPropertyChanged("LimitMax"); }
}

#region Constructor
//public event Action CommitEditBeforeSave;
//public event Action ChangeOccured;

public override void PaneClose()
{
this.CalLimits.CommitEditBeforeSave -= View.ForceCommitEdit;
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
if (this.CalLimits != null)
{
this.CalLimits.FilePath = value;
}
}
}

public LimitUcVM(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
//this.CalLimits = (CalLimitsModel)dataObject;
```

```
//
////Temp. Workaround for future backwards compatibility purposes
//DataFilePath = dataObject.FilePath;

this.DataFilePath = dataFilePath;

this.LoadData();

this.CalLimits.ChangeOccured += ChangeOccuredSubscriber;
}

public override void SaveData()
{
ForceCommitEdit();
//CommitEditBeforeSave?.Invoke();

//this.CalLimits.Save_Custom_XML(this.DataFilePath);
this.CalLimits.SaveToXml(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public void ForceCommitEdit()
{
if (View != null)
{
View.ForceCommitEdit();
}
}

public override void LoadData()
{
CalLimitsModel dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = CalLimitsModel.LoadFromXml(this.DataFilePath);
}

if (dataModel != null)
{
```

```
this.CalLimits = dataModel;

//IFileData stuff
dataModel.FilePath = this.DataFilePath;

this.ParentProject.CalibrationLimits.Add(dataModel);
}

base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{

}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
//public void OnChangeOccured()
//{
//    ChangeOccured?.Invoke();
//}

#endregion

#region Commands

#region Autogenerate Cal Limits
public ICommand AutoGenerateCalLimitsCommand { get { return new
DelegateCommand(AutoGenerateCalLimits); } }
private void AutoGenerateCalLimits(object obj)
{
CalLimits.SetLimitsTolerance(SelectedCalResult, LimitMin, LimitMax);
CalLimits.OnChangeOccured();
}
#endregion Autogenerate Cal Limits

#endregion

}
}
```

ResultDataVM.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public class ResultDataVM : PVM
{
private List<object> Data = new List<object>();
public string resultParentName = string.Empty;
public string resultName = string.Empty;



//Temp
private List<string> _unitList = new List<string>() { "Hz", "KHz", "MHz", "GHz" };
//Temp.
public IEnumerable<string> UnitList { get { return _unitList; } }

private string _viewingUnit = "GHz";
public string ViewingUnit
{
get { return _viewingUnit; }
set { _viewingUnit = value; OnPropertyChanged("ViewingUnit"); }
}



//Data accessor for Cal Result data bindings.
public List<object> GridViewData
{
get { return Data; }
}

//Data accessor for Cal Def Info data bindings and Model parent matching.
public CalDataModel CalDataInfo { get; set; }

#region Constructors
public override void PaneClose()
{
```

```
//this.View = null; //No View to nullify. Already unloads UC on pane close.
}

//Constructor for Cal Result data viewing
public ResultDataVM(Type contentType, CalResult calResult, CalDataModel calDataInfo) :
base(contentType)
{
resultParentName = calResult.CalDataModelParent;
resultName = calResult.ResultName;
Data = calResult.KvPMergedObjectData;
this.CalDataInfo = calDataInfo; //Set parent. used for pane removal.
}

//Constructor for Cal Data Model Info viewing.
public ResultDataVM(Type contentType, CalDataModel calDataInfo) : base(contentType)
{
this.CalDataInfo = calDataInfo; //Set parent, used for data viewing and pane removal.
}
#endregion Constructors

}
}
```

ProductConfigViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ComponentModel;
using System.Windows;
using MerlinTest.Tools.TestStudio.Model;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio.DataModels;
using System.IO;

namespace MT.TestStudio.GUI.ViewModels
{
public interface IProductConfigView
{
```

```
}

/// <summary>
/// ViewModel for the Product Configuration UserControl.
/// </summary>
public class ProductConfigViewModel : PVM
{
private IProductConfigView _view;
public IProductConfigView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");
if (value != null)
{
//View.RebindTestLimitsGridView();
//this.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}

#region Private Member Variables

private ProductConfiguration _modelObj = new ProductConfiguration();
private bool continueOnFail;
private bool stopOnFail;
private bool continueOnAllFail;
private bool stopOnalarm;
private bool continueOnAlarm;
private bool goldUnitEnable;
private bool userCalibrationEnabled;
private int calibrationExpiration;
private bool offlineQaAvailable;
private bool encryptRequired;
private int inlineNthDevice;
private int logNthDevice;
private int numberOfsites;

#endregion Private Member Variables
```

```
public ProductConfiguration ProductConfigurationData
{
get { return _modelObj; }
set { _modelObj = value; OnPropertyChanged("ProductConfigurationData"); }
}

#region Constructor
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;

public override void PaneClose()
{
//this.CommitEditBeforeSave -= View.ForceCommitEdit;
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
}
}

/// <summary>
/// Constructor for the Hardware Configuration View Model.
/// </summary>
public ProductConfigViewModel(Type contentType, string dataFilePath, MerlinProject
parentProject, ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject,
parentFolder)
{
//// Don't execute the viewmodel code if we're just looking at the XAML in the designer.
//if (DesignerProperties.GetIsInDesignMode(new DependencyObject())) return;

this.DataFilePath = dataFilePath;

this.LoadData();
```

```
this.ChangeOccured += ChangeOccuredSubscriber;
}

public override void SaveData()
{
//CommitEditBeforeSave?.Invoke();

ProductConfigurationData.StopOnFail = this.StopOnFail;
ProductConfigurationData.StopOnalarm = this.StopOnalarm;
ProductConfigurationData.ContinueOnFail = this.ContinueOnFail;
ProductConfigurationData.ContinueOnAllFail = this.ContinueOnAllFail;
ProductConfigurationData.ContinueOnAlarm = this.ContinueOnAlarm;
ProductConfigurationData.GoldUnitEnable = this.GoldUnitEnable;
ProductConfigurationData.UserCalibrationEnable = this.UserCalibrationEnabled;
ProductConfigurationData.CalibrationExpiration = this.CalibrationExpiration;
ProductConfigurationData.OfflineQaAvailable = this.OfflineQaAvailable;
ProductConfigurationData.EncryptRequired = this.EncryptRequired;
ProductConfigurationData.InlineNthDevice = this.InlineNthDevice;
ProductConfigurationData.LogNthDevice = this.LogNthDevice;
ProductConfigurationData.NumberOfsites = this.NumberOfsites;

this.ProductConfigurationData.SaveToXml(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public override void LoadData()
{
ProductConfiguration dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = ProductConfiguration.LoadFromXml(this.DataFilePath);
}

if (dataModel != null)
{
this.ProductConfigurationData = dataModel;

this.stopOnFail = ProductConfigurationData.StopOnFail; OnPropertyChanged("StopOnFail");
this.stopOnalarm = ProductConfigurationData.StopOnalarm;
```

```
OnPropertyChanged("StopOnalarm");
this.continueOnFail = ProductConfigurationData.ContinueOnFail;
OnPropertyChanged("ContinueOnFail");
this.continueOnAllFail = ProductConfigurationData.ContinueOnAllFail;
OnPropertyChanged("ContinueOnAllFail");
this.continueOnAlarm = ProductConfigurationData.ContinueOnAlarm;
OnPropertyChanged("ContinueOnAlarm");
this.goldUnitEnable = ProductConfigurationData.GoldUnitEnable;
OnPropertyChanged("GoldUnitEnable");
this.userCalibrationEnabled = ProductConfigurationData.UserCalibrationEnable;
OnPropertyChanged("UserCalibrationEnabled");
this.calibrationExpiration = ProductConfigurationData.CalibrationExpiration;
OnPropertyChanged("CalibrationExpiration");
this.offlineQaAvailable = ProductConfigurationData.OfflineQaAvailable;
OnPropertyChanged("OfflineQaAvailable");
this.encryptRequired = ProductConfigurationData.EncryptRequired;
OnPropertyChanged("EncryptRequired");
this.inlineNthDevice = ProductConfigurationData.InlineNthDevice;
OnPropertyChanged("InlineNthDevice");
this.logNthDevice = ProductConfigurationData.LogNthDevice;
OnPropertyChanged("LogNthDevice");
this.numberOfsites = ProductConfigurationData.NumberOfsites;
OnPropertyChanged("NumberOfsites");
}

base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{

}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}

#endregion Constructor

#region Public Properties
```

```
/// <summary>
/// Gets / Sets if the test flow should continue on Fail.
/// </summary>
public bool ContinueOnFail
{
get { return stopOnFail; }
set
{
continueOnFail = value;
OnChangeOccured();
OnPropertyChanged("ContinueOnFail");
}
}

/// <summary>
/// Gets / Sets if the test flow should stop on Fail.
/// </summary>
public bool StopOnFail
{
get { return stopOnFail; }
set
{
stopOnFail = value;
OnChangeOccured();
OnPropertyChanged("StopOnFail");
}
}

/// <summary>
/// Gets / Sets if the test flow should continue if all tests fail.
/// </summary>
public bool ContinueOnAllFail
{
get { return continueOnAllFail; }
set
{
continueOnAllFail = value;
OnChangeOccured();
OnPropertyChanged("ContinueOnAllFail");
}
}
```

```
/// <summary>
/// Gets / Sets if the test flow should stop on an Alarm.
/// </summary>
public bool StopOnalarm
{
get { return stopOnalarm; }
set
{
stopOnalarm = value;
OnChangeOccured();
OnPropertyChanged("StopOnalarm");
}
}

/// <summary>
/// Gets / Sets if the test flow should continue on an Alarm.
/// </summary>
public bool ContinueOnAlarm
{
get { return continueOnAlarm; }
set
{
continueOnAlarm = value;
OnChangeOccured();
OnPropertyChanged("ContinueOnAlarm");
}
}

/// <summary>
/// Gets / Sets if the test flow is/should executing a GOLDen Unit.
/// </summary>
public bool GoldUnitEnable
{
get { return goldUnitEnable; }
set
{
goldUnitEnable = value;
OnChangeOccured();
OnPropertyChanged("GoldUnitEnable");
}
}
```

```
/// <summary>
/// Gets / Sets if the test flow should perform a User Calibration.
/// </summary>
public bool UserCalibrationEnabled
{
get { return userCalibrationEnabled; }
set
{
userCalibrationEnabled = value;
OnChangeOccured();
OnPropertyChanged("UserCalibrationEnabled");
}
}

/// <summary>
/// Gets / Sets the number of days before the user calibration is considered expired.
/// </summary>
public int CalibrationExpiration
{
get { return calibrationExpiration; }
set
{
calibrationExpiration = value;
OnChangeOccured();
OnPropertyChanged("CalibrationExpiration");
}
}

/// <summary>
/// Gets / Sets if offline QA is available.
/// </summary>
public bool OfflineQaAvailable
{
get { return offlineQaAvailable; }
set
{
offlineQaAvailable = value;
OnChangeOccured();
OnPropertyChanged("OfflineQaAvailable");
}
}
```

```
/// <summary>
/// Gets / Sets if encryption is required.
/// </summary>
public bool EncryptRequired
{
get { return encryptRequired; }
set
{
encryptRequired = value;
OnChangeOccured();
OnPropertyChanged("EncryptRequired");
}
}

/// <summary>
/// Gets / Sets the value of the nth device that will be QA tested.
/// </summary>
public int InlineNthDevice
{
get { return inlineNthDevice; }
set
{
inlineNthDevice = value;
OnChangeOccured();
OnPropertyChanged("InlineNthDevice");
}
}

/// <summary>
/// Gets / Sets the value of the nth device that will be logged.
/// </summary>
public int LogNthDevice
{
get { return logNthDevice; }
set
{
logNthDevice = value;
OnChangeOccured();
OnPropertyChanged("LogNthDevice");
}
}
```

```
/// <summary>
/// Gets / Sets the number of sites to be tested.
/// </summary>
public int NumberOfsites
{
get { return numberOfsites; }
set
{
numberOfsites = value;
OnChangeOccured();
OnPropertyChanged("NumberOfsites");
}
}

#endregion Public Properties



}
}
```

ProjectConfigViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ComponentModel;
using System.Windows;
using MerlinTest.Tools.TestStudio.Model;
using MerlinTestStudio_Demo_Telerik;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Runtime.Serialization.Formatters.Binary;
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;

namespace MT.TestStudio.GUI.ViewModels
{
/// <summary>
/// ViewModel for the Project Configuration UserControl.
/// </summary>
public class ProjectConfigViewModel : UcViewModelBase, INotifyPropertyChanged, IPVMLink
```

```
{
private PaneViewModel _pvm;
public PaneViewModel PVM
{
get { return _pvm; }
set
{
_pvm = value; OnPropertyChanged("PVM");
}
}
#region Private Member Variables

private string _userTargetSystem;

/// <summary>
/// Instance of the Model Object.
/// </summary>
private ProjectConfigModel ModelObj { get; set; }

/// <summary>
/// Gets / Sets the project filename.
/// </summary>
private string projectFileName;

/// <summary>
/// Gets / Sets the project file version..
/// </summary>
private int projectFileVersion;

/// <summary>
/// Gets / Sets the project file path..
/// </summary>
private string projectFilePath;

/// <summary>
/// Gets / Sets the project name.
/// </summary>
private string projectName;

/// <summary>
/// Gets / Sets the project name.
/// </summary>
```

```
private string productName;

/// <summary>
/// Gets / Sets the project date.
/// </summary>
private DateTime projectDate;

/// <summary>
/// Gets / Sets the project comment.
/// </summary>
private string projectComment;

#endregion Private Member Variables

public IEnumerable<string> TargetSystemOptions { get { return
BackendConstants.TargetSystemOptions; } }

#region Constructor
/// <summary>
/// Constructor for the Hardware Configuration View Model.
/// </summary>
public ProjectConfigViewModel()
{
// Don't execute the viewmodel code if we're just looking at the XAML in the designer.
if (DesignerProperties.GetIsInDesignMode(new DependencyObject())) return;

// Create an instance of the Calibration Model class.
ModelObj = new ProjectConfigModel();

// Set Some Defaults
this.IsDataLoaded = false;
}

#endregion Constructor

#region Public Properties

//public string UserTargetSystem
//{
//    get { return PVM.ParentProject.UserTargetSystem; }
//    set
//    {
```

```
//        PVM.ParentProject.UserTargetSystem = value;
//        OnPropertyChanged("UserTargetSystem");
//
//        //Then loop through the CalDefs to set the current UserTargetSystem value.
//        foreach(CalDataModel cdm in PVM.ParentProject.CalibrationDefinitions)
//        {
//            cdm.GlobalSystemVarient = value;
//        }
//
//        PVM.IsSaveRequired = true;
//    }
//}
//
//public string ProjectFileName
//{
//    get
//    {
//        return PVM.ParentProject.ProjectFileName;
//    }
//    //set
//    //{
//    //    PVM.ParentProject.ProjectFileName = value;
//    //    // Update the Save Required property now that the data has changed.
//    //    PVM.IsSaveRequired = true;
//    //    OnPropertyChanged("ProjectFileName");
//    //}
//}
//
//public double ProjectFileVersion
//{
//    get
//    {
//        return PVM.ParentProject.Version;
//    }
//    //set
//    //{
//    //    projectFileVersion = value;
//    //    // Update the Save Required property now that the data has changed.
//    //    PVM.IsSaveRequired = true;
//    //    OnPropertyChanged("ProjectFileName");
//    //}
//}
```

```
//
//public string ProjectFilePath
//{
//    get
//    {
//        return PVM.ParentProject.ProjectFilePath;
//    }
//    //set
//    //{
//    //    projectFilePath = value;
//    //    // Update the Save Required property now that the data has changed.
//    //    PVM.IsSaveRequired = true;
//    //    OnPropertyChanged("ProjectFilePath");
//    //}
//}
//
//public string ProjectName
//{
//    get
//    {
//        return PVM.ParentProject.ProjectName;
//    }
//    //set
//    //{
//    //    projectName = value;
//    //    // Update the Save Required property now that the data has changed.
//    //    PVM.IsSaveRequired = true;
//    //    OnPropertyChanged("ProjectName");
//    //}
//}
//
//public string ProductName
//{
//    get
//    {
//        return PVM.ParentProject.ProductName;
//    }
//    //set
//    //{
//    //    productName = value;
//    //    // Update the Save Required property now that the data has changed.
//    //    PVM.IsSaveRequired = true;
```

```
//    //    OnPropertyChanged("ProductName");
//    //}
//}
//
//public DateTime ProjectDate
//{
//    get
//    {
//        return projectDate;
//    }
//    set
//    {
//        projectDate = value ;
//        // Update the Save Required property now that the data has changed.
//        PVM.IsSaveRequired = true;
//        OnPropertyChanged("ProjectDate");
//    }
//}
//
//public string ProjectComment
//{
//    get
//    {
//        return projectComment;
//    }
//    set
//    {
//        projectComment = value;
//        // Update the Save Required property now that the data has changed.
//        PVM.IsSaveRequired = true;
//        OnPropertyChanged("ProjectComment");
//    }
//}

#endregion Public Properties

#region Public Methods

/// <summary>
/// Method that causes the file associated with this instance of the Project settings viewModel to load.
/// </summary>
```

```
public override void LoadDataFile()
{
// Only load the data if it's not been loaded already.
if (IsDataLoaded == false)
{
this.ModelObj.LoadCsvDataFile(this.FileName);

this.IsDataLoaded = true;
}
}

/// <summary>
/// Method that causes the data in the view-model to be written to the file associated with this
instance of the Project Setting viewModel.
/// </summary>
public override void SaveDataFile()
{
////// Push All View-Model values to GUI.
/////
//ModelObj.ProjectName = this.ProjectName;
//ModelObj.ProjectFilePath = this.ProjectFilePath;
//ModelObj.ProjectFileName = this.ProjectFileName;
//ModelObj.ProjectFileVersion = this.ProjectFileVersion;
//ModelObj.ProductName = this.ProductName;
//ModelObj.ProjectDate = this.ProjectDate;
//ModelObj.ProjectComment = this.ProjectComment;
//ModelObj.UserTargetSystem = this.UserTargetSystem;
//
//// Now Save Data in Model to disk.
//this.ModelObj.SaveCsvDataFile(this.FileName);
//
//// Indicate that the file no longer needs saved.
//this.SaveRequired = false;
}

/// <summary>
/// Method that gets called whenever the Model has updated a value.
/// </summary>
/// <param name="sender"></param>
/// <param name="e">Event argument that contains the property name that has
changed.</param>
private void ModelObj_PropertyChanged(object sender, PropertyChangedEventArgs e)
```

```
{
//if (e.PropertyName == "CsvDataLoaded")
//{
//    // Update the properties. However do it to the private copies and then call OnproprtyChanged. This has the effect of
//    // updating the GUI but doesn't call the SET part of each property. ( which we use for detecting if the data has changed )
//    this.projectName = ModelObj.ProjectName;
//    this.productName = ModelObj.ProductName;
//    this.projectDate = ModelObj.ProjectDate;
//    this.projectComment = ModelObj.ProjectComment;
//    this.projectFileVersion = ModelObj.ProjectFileVersion;
//    this.projectFileName = ModelObj.ProjectFileName;
//    this.projectFilePath = ModelObj.ProjectFilePath;
//    this._userTargetSystem = ModelObj.UserTargetSystem;
//
//    OnPropertyChanged("ProjectName");
//    OnPropertyChanged("ProductName");
//    OnPropertyChanged("ProjectDate");
//    OnPropertyChanged("ProjectComment");
//    OnPropertyChanged("ProjectFileVersion");
//    OnPropertyChanged("ProjectFileName");
//    OnPropertyChanged("ProjectFilePath");
//    OnPropertyChanged("UserTargetSystem");
//}
}

#endregion Public Methods
}
}
```

SystemConfigViewModel.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Services;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.ObjectModel;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Windows;
using System.Windows.Input;
```

```
using System.Xml.Serialization;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public class SystemConfigViewModel : UcViewModelBase, IPVMLink
{
private PaneViewModel _pvm;
public PaneViewModel PVM
{
get { return _pvm; }
set { _pvm = value; OnPropertyChanged("PVM"); this.Instruments =
PVM.MVM.MySystem.GetInstruments(); }
}
#region Public & Private variables

private ObservableCollection<Instrument> _instruments = new
ObservableCollection<Instrument>();
public ObservableCollection<Instrument> Instruments
{
get { return _instruments; } set { _instruments = value; OnPropertyChanged("Instruments"); }
}

//// Represents the selected instrument list.
//private string _instrumentItemSelected;
//public string InstrumentItemSelected
//{
//    get { return _instrumentItemSelected; }
//    set
//    {
//        if (value != _instrumentItemSelected)
//        {
//            _instrumentItemSelected = value;
//            OnPropertyChanged(() => _instrumentItemSelected);
//        }
//    }
//}

//Instrument Properties Bindings
public string instrumentName { get; set; }
public int instrumentID { get; set; }
public string serialNumber { get; set; }
```

```
public ObservableCollection<string> ports { get; set; }

// Represents the selected instrument.
private Data.Instrument _Selected;
public Data.Instrument Selected
{
get { return _Selected; }
set
{
if (value != _Selected)
{
_Selected = value;
OnPropertyChanged("Selected");
if(value != null)
{
instrumentName = value.InstrumentName; OnPropertyChanged("instrumentName");
//instrumentID = value.InstrumentID; OnPropertyChanged("instrumentID");
serialNumber = value.SerialNumber; OnPropertyChanged("serialNumber");
ports = value.Ports; OnPropertyChanged("ports");
}
}
}
}

#endregion

IInstrumentService _instrumentService;
IProjectConfigurationService _configurationService;
IViewModelLocator _viewModelLocator;
public event Action<string> OnChangesMade = (UserControl) => { };

/// <summary>
/// Constructor for the SystemConfigViewModel.
/// </summary>
public SystemConfigViewModel(IInstrumentService instrumentService,
IProjectConfigurationService configurationService,
IViewModelLocator viewModelLocator)
{
FileName = "System Configuration";

_instrumentService = instrumentService;
_configurationService = configurationService;
```

```
_viewModelLocator = viewModelLocator;
OnChangesMade += (UserControl) =>
{
PVM.IsSaveRequired = true;

//_viewModelLocator.OnControlInteract(UserControl);
//SaveRequired = true;
};

_OpenContent = new DelegateCommand(OpenContent, CanOpenContent);
_AddInstrument = new DelegateCommand(AddInstrument);
}

public override void SaveDataFile()
{
base.SaveDataFile();
//string projectFilePath = _viewModelLocator.GetProjectFilePath();
//BinaryFormatter bf = new BinaryFormatter();
//PVM.ParentProject.SaveInstrumentsData(ref bf, projectFilePath);

SaveRequired = false;
}


#region Commands

private readonly DelegateCommand _OpenContent;
public ICommand OpenContentCommand => _OpenContent;

private readonly DelegateCommand _AddInstrument;
public ICommand AddInstrumentCommand => _AddInstrument;

#endregion

#region Command Methods

private void OpenContent(object commandSender)
{

}
private bool CanOpenContent(object commandSender)
```

```
{
return true;
}

public ICommand DeleteInstrumentCommand { get { return new
DelegateCommand(DeleteInstrument); } }
private void DeleteInstrument(object obj)
{
if(Selected != null)
{
switch (Selected.InstrumentType)
{
case MTSInstrumentType.RF_Instrument:
PVM.MVM.MySystem.RF_Instruments.Remove(Selected);
break;
case MTSInstrumentType.PXI:
PVM.MVM.MySystem.PXI_Instruments.Remove(Selected);
break;
case MTSInstrumentType.PowerSupply:
PVM.MVM.MySystem.Power_Instruments.Remove(Selected);
break;
}

this.Instruments = PVM.MVM.MySystem.GetInstruments();
}
}

public ICommand ResetSystemCommand { get { return new DelegateCommand(ResetSystem); } }
private void ResetSystem(object obj)
{
PVM.MVM.MySystem.ClearInstruments();
PVM.MVM.MySystem.RF_Instruments.Add(new
MerlinTestStudio_Demo_Telerik.Data.Instrument() { InstrumentName = "RF110 (1)",
InstrumentType = MTSInstrumentType.RF_Instrument });
PVM.MVM.MySystem.RF_Instruments.Add(new
MerlinTestStudio_Demo_Telerik.Data.Instrument() { InstrumentName = "RF210 (1)",
InstrumentType = MTSInstrumentType.RF_Instrument });
PVM.MVM.MySystem.PXI_Instruments.Add(new
MerlinTestStudio_Demo_Telerik.Data.Instrument() { InstrumentName = "PE32H (1)",
InstrumentType = MTSInstrumentType.PXI });
PVM.MVM.MySystem.PXI_Instruments.Add(new
MerlinTestStudio_Demo_Telerik.Data.Instrument() { InstrumentName = "PE32H (2)",
```

```
InstrumentType = MTSInstrumentType.PXI });
PVM.MVM.MySystem.PXI_Instruments.Add(new
MerlinTestStudio_Demo_Telerik.Data.Instrument() { InstrumentName = "SEDPin32 (1)",
InstrumentType = MTSInstrumentType.PXI });
PVM.MVM.MySystem.PXI_Instruments.Add(new
MerlinTestStudio_Demo_Telerik.Data.Instrument() { InstrumentName = "SEDPin32 (2)",
InstrumentType = MTSInstrumentType.PXI });
PVM.MVM.MySystem.Power_Instruments.Add(new
MerlinTestStudio_Demo_Telerik.Data.Instrument() { InstrumentName = "PSM (1)",
InstrumentType = MTSInstrumentType.PowerSupply });
this.Instruments = PVM.MVM.MySystem.GetInstruments();
}

private void AddInstrument(object commandSender)
{
string instrumentStr = commandSender.ToString();
if (instrumentStr != null)
{
Instrument instrumentToAdd = new Instrument();
//instrumentToAdd.InstrumentID = PVM.MVM.MySystem.RF_Instruments.Count +
PVM.MVM.MySystem.PXI_Instruments.Count + PVM.MVM.MySystem.Power_Instruments.Count;
switch (instrumentStr)
{
case "DPS":
case "PSM":
case "HPA100":
instrumentToAdd.InstrumentName =
GetUniqueInstrumentName(PVM.MVM.MySystem.Power_Instruments, instrumentStr);
instrumentToAdd.InstrumentType = MTSInstrumentType.PowerSupply;
PVM.MVM.MySystem.Power_Instruments.Add(instrumentToAdd);
break;
case "RF100":
case "RF110":
case "RF200":
case "RF210":
case "RF300":
instrumentToAdd.InstrumentName =
GetUniqueInstrumentName(PVM.MVM.MySystem.RF_Instruments, instrumentStr);
instrumentToAdd.InstrumentType = MTSInstrumentType.RF_Instrument;
PVM.MVM.MySystem.RF_Instruments.Add(instrumentToAdd);
break;
case "OpenATE":
```

```
case "PE32H":
case "SEDPin32":
instrumentToAdd.InstrumentName =
GetUniqueInstrumentName(PVM.MVM.MySystem.PXI_Instruments, instrumentStr);
instrumentToAdd.InstrumentType = MTSInstrumentType.PXI;
PVM.MVM.MySystem.PXI_Instruments.Add(instrumentToAdd);
break;
default: break;
}

this.Instruments = PVM.MVM.MySystem.GetInstruments();
//OnChangesMade("System Configuration");
}

}
private string GetUniqueInstrumentName(ObservableCollection<Instrument> InstCollection, string
instrumentName)
{
int iteration = 1;
bool uniqueName;
string newUniqueName = string.Format("{0} (1)", instrumentName);
// Iterate through all items in list looking to see if the name matches the newFile argument.
// If not then just return the NewFile argument otherwise create a new filename based on NewFile
// and test to see if it's used and repeat until a unique filename is used.
do
{
uniqueName = true;

foreach (Instrument inst in InstCollection)
{
if (inst.InstrumentName == newUniqueName)
{
uniqueName = false;
// File Name is being used and so create a new fileName.
iteration++;
newUniqueName = string.Format("{0} ({1})", instrumentName, iteration);
break;
}
}
// Test to see if we've found a unique name.
} while (uniqueName != true);
```

```
return newUniqueName;
}

#endregion CommandMethods
}
}
```

ConnectionDiagramViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ComponentModel;
using System.Windows;
using MerlinTest.Tools.TestStudio.Model;
using System.Windows.Data;
using System.Windows.Media.Imaging;

namespace MT.TestStudio.GUI.ViewModels
{
/// <summary>
/// ViewModel for the connection diagram UserControl.
/// </summary>
public class ConnectionDiagramViewModel : UcViewModelBase, INotifyPropertyChanged
{
#region Private Member Variables

#endregion Private Member Variables

#region Constructor

/// <summary>
/// Constructor for the Hardware Configuration View Model.
/// </summary>
public ConnectionDiagramViewModel()
{
// Don't execute the viewmodel code if we're just looking at the XAML in the designer.
if (DesignerProperties.GetIsInDesignMode(new DependencyObject())) return;

// Set Some Defaults
this.IsDataLoaded = false;
```

```
}

#endregion Constructor

#region Public Properties


/// <summary>
/// The filename of the image to be displayed.
/// </summary>
public string DisplayedImage
{
get
{
return this.FileName;
}
set
{
this.FileName = value;
OnPropertyChanged("DisplayedImage");
}
}

#endregion Public Properties

#region Public Methods

/// <summary>
/// Method that causes the file associated with this instance of the Connection Diagram viewModel
to loaded.
/// </summary>
public override void LoadDataFile()
{
// Only load the data if it's not been loaded already.
if (IsDataLoaded == false)
{
// this.ModelObj.LoadCsvDataFile(this.FileName);
OnPropertyChanged("DisplayedImage");

this.IsDataLoaded = true;
}
}
```

```
/// <summary>
/// Method that gets called whenever the Model has updated a value.
/// </summary>
/// <param name="sender"></param>
/// <param name="e">Event argument that contains the property name that has
changed.</param>
private void ModelObj_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
if (e.PropertyName == "CsvDataLoaded")
{
// Do nothing as we're just loading a static image.
}
}

#endregion Public Methods
}
}
```

PinMapViewModel.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Converters;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.PinModels;
using MerlinTestStudio_Demo_Telerik.Data.Services;
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Data;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.Windows;
using System.Windows.Data;
using System.Windows.Input;
using System.Windows.Markup;
using System.Xml.Serialization;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
```

```
public interface IPinMapView
{
/// <summary>
/// Clears the focus on the GridView respresenting PinMap Editor.
/// </summary>
void UnselectAll();

/// <summary>
///Fire when any of the save/export commands are fired, this Forces the grid to commit any pending edits from cells in edit mode.
/// </summary>
void ForceCommitEdit();

//Temporarily gets the rad grid view from the user control to support all of the old code.
RadGridView GetGridView();

}

[XmlInclude(typeof(PinMapViewModel))]
public class PinMapViewModel : PVM //IPVMLink
{
//private PaneViewModel _pvm;
//[XmlIgnore]
//public PaneViewModel PVM
//{
//    get { return _pvm; }
//    set
//    {
//        _pvm = value;
//        OnPropertyChanged("PVM");
//        //this.CurrentPinMapModel = (PinMapModel)value.DataObject;
//        //CurrentPinMapModel.SiteRemoved += RemoveSiteGUI; //Subscribe to the event.
//        //CurrentPinMapModel.SiteAdded += CreateSiteGUI; //Subscribe to the event.
//        //CurrentPinMapModel.CommitEditBeforeSave += View.ForceCommitEdit;
//        //RebindPinMapGridView();
//    }
//}
[XmlIgnore]
public static string UserDialogEntry { get; set; }

private IPinMapView _view;
[XmlIgnore]
```

```
public IPinMapView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");
//DataObject should already be set in the constructor.
if (value != null)
{
CurrentPinMapModel.SiteRemoved += RemoveSiteGUI; //Subscribe to the event.
CurrentPinMapModel.SiteAdded += CreateSiteGUI; //Subscribe to the event.
CurrentPinMapModel.CommitEditBeforeSave += View.ForceCommitEdit;
RebindPinMapGridView();
}
}
}

private ObservableCollection<string> _ImportPinsCollection = new
ObservableCollection<string>();
private ObservableCollection<string> _AvailablePinsCollection = new
ObservableCollection<string>();
private ObservableCollection<string> _selectedImportPin = new ObservableCollection<string>();
private ObservableCollection<string> _selectedAvailablePin = new
ObservableCollection<string>();
private Site _selectedSite;
private static Style PinMapGridViewRowStyle =
Application.Current.FindResource("PinMapGridViewRowStyle") as Style;

private PinMapModel _currentPinMapModel = new PinMapModel();
[XmlIgnore]
public PinMapModel CurrentPinMapModel
{
get { return _currentPinMapModel; } set { _currentPinMapModel = value;
OnPropertyChanged("CurrentPinMapModel"); }
}
[XmlIgnore]
public Site SelectedSite
{
get { return _selectedSite; }
set { _selectedSite = value; OnPropertyChanged("SelectedSite"); }
}
```

```
[XmlIgnore]
public ObservableCollection<Site> ActiveSites
{
get
{
ObservableCollection<Site> activeSites = new ObservableCollection<Site>();
for (int i = 0; i <= MaxMappingsCount; i++) //Collects currently open sites.
{
if (i > 0)
{
activeSites.Add(new Site() { SiteID = i, SiteName = "Site " + i });
}
}
return activeSites;
}
}

//SWITCH to the one in the pin map model.
[XmlIgnore]
public int MaxMappingsCount
{
get
{
if (CurrentPinMapModel.RfPins.Count == 0)
{
if(CurrentPinMapModel.DigitalPins.Count() == 0)
{
if(CurrentPinMapModel.PowerSupplyPins.Count() == 0)
{
return 0;
}
else { return CurrentPinMapModel.PowerSupplyPins.Max((x) => x.mappings.Count); }
}
else { return CurrentPinMapModel.DigitalPins.Max((x) => x.mappings.Count); }
}
else
{ return CurrentPinMapModel.RfPins.Max((x) => x.mappings.Count); }
}
}

[XmlIgnore]
public ObservableCollection<string> ImportPinsCollection
```

```
{
get { return _ImportPinsCollection; }
set
{
_ImportPinsCollection = value;
OnPropertyChanged("ImportPinsCollection");
}
}
[XmlIgnore]
public ObservableCollection<string> SelectedImportPin
{
get { return _selectedImportPin; }
set
{
_selectedImportPin = value;
OnPropertyChanged("SelectedImportPin");
}
}
[XmlIgnore]
public ObservableCollection<string> AvailablePinsCollection
{
get { return _AvailablePinsCollection; }
set
{
_AvailablePinsCollection = value;
OnPropertyChanged("AvailablePinsCollection");
}
}
[XmlIgnore]
public ObservableCollection<string> SelectedAvailablePin
{
get { return _selectedAvailablePin; }
set
{
_selectedAvailablePin = value;
OnPropertyChanged("SelectedAvailablePin");
}
}

[XmlIgnore]
public static RadWindow ImportPinWindow = new RadWindow
{
```

```
Width = 800,
Height = 600,
WindowStartupLocation = System.Windows.WindowStartupLocation.CenterScreen,
Content = new UserControls.ImportPinsWindowContent(),
Header = "Import Pins"
};

private string _selectedPinMapSection = "RF Pins" ;
[XmlIgnore]
public string SelectedPinMapSection
{
get { return _selectedPinMapSection; }
set
{
_selectedPinMapSection = value;
if(PinMapGrid != null)
{
//Go through and set selected item source and the instrument column item sources.
ObservableCollection<Data.Instrument> instrumentsSource = null;
switch (PinMapModel.GetPinMapSectionEnum(value))
{
case PinMapSection.Rf_Pins:
PinMapGrid.ItemsSource = CurrentPinMapModel.RfPins;
instrumentsSource = MVM.MySystem.RF_Instruments;
break;
case PinMapSection.Digital_Pins:
PinMapGrid.ItemsSource = CurrentPinMapModel.DigitalPins;
instrumentsSource = MVM.MySystem.PXI_Instruments;
break;
case PinMapSection.Power_Supply_Pins:
PinMapGrid.ItemsSource = CurrentPinMapModel.PowerSupplyPins;
instrumentsSource = MVM.MySystem.Power_Instruments;
break;
}
foreach (GridViewColumn gvc in PinMapGrid.Columns) //Set the instrument binding source based
on pin map section.
{
if (gvc.Header.ToString() == "Instrument: ")
{
(gvc as GridViewComboBoxColumn).ItemsSource = instrumentsSource;
}
}
```

```
}
}
}

[XmlIgnore]
public RadGridView PinMapGrid { get { return View.GetGridView(); } }

private void RebindPinMapGridView()
{
try
{
ObservableCollection<Data.PinModel> pinSource = CurrentPinMapModel.RfPins; //Use RF pins as the default for load in itemsource (also for new projects no source bug)
ObservableCollection<Data.Instrument> instrumentSource = null;
switch (SelectedPinMapSection)
{
case "RF Pins":
pinSource = CurrentPinMapModel.RfPins;
instrumentSource = MVM.MySystem.RF_Instruments;
break;
case "Digital Pins":
pinSource = CurrentPinMapModel.DigitalPins;
instrumentSource = MVM.MySystem.PXI_Instruments;
break;
case "Power Supply Pins":
pinSource = CurrentPinMapModel.PowerSupplyPins;
instrumentSource = MVM.MySystem.Power_Instruments;
break;
}
//Sets the itemsource of the PinMapGridView.
PinMapGrid.ItemsSource = pinSource;

PinMapGrid.ColumnGroups.Add(new GridViewColumnGroup() { Name = "PinInfo", Header = new System.Windows.Controls.TextBlock() { Text = "PIN INFO", VerticalAlignment = VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } });
//Add the known columns
PinMapGrid.Columns.Add(new GridViewDataColumn()
{
UniqueName = "PinName",
Header = "Pin Name: ",
DataMemberBinding = new Binding("PinName"),
ColumnGroupName = "PinInfo",
```

```
MinWidth = 100
});
PinMapGrid.Columns.Add(new GridViewDataColumn()
{
UniqueName = "PinAlias",
Header = "Pin Alias: ",
DataMemberBinding = new Binding("PinAlias"),
ColumnGroupName = "PinInfo",
MinWidth = 100
});
//Creates Sites for Pins
if (CurrentPinMapModel.RfPins.Count != 0 || CurrentPinMapModel.DigitalPins.Count != 0 ||
CurrentPinMapModel.PowerSupplyPins.Count != 0)
{
//int maxSitesNumber = ParentProject.Pins.Max((x) => x.mappings.Count);
for (int i = 0; i < MaxMappingsCount; i++)
{
PinMapGrid.ColumnGroups.Add(new GridViewColumnGroup() { Name = ("Site " + (i + 1)), Header
= new System.Windows.Controls.TextBlock() { Text = "SITE " + (i +1), VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } });

string xaml = @"
<DataTemplate xmlns=""http://schemas.microsoft.com/winfx/2006/xaml/presentation"">
<TextBlock Foreground=""{Binding Path=mappings[" + i + @"].Instrument, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource NullToColorConverter}}"">
<TextBlock.Text>
<PriorityBinding>
<Binding Path=""mappings[" + i + @"].InstrumentName"" Mode=""OneWay""
UpdateSourceTrigger=""PropertyChanged"" IsAsync=""True"" />
<Binding Path=""InstrumentName"" Mode=""OneWay""
UpdateSourceTrigger=""PropertyChanged"" IsAsync=""True"" />
</PriorityBinding>
</TextBlock.Text>
</TextBlock>
</DataTemplate>
";
DataTemplate cellTemplate = (DataTemplate)XamlReader.Parse(xaml);

// Define the CellStyle
Style cellStyle = new Style(typeof(GridViewCell));
DataTrigger dataTrigger = new DataTrigger();
dataTrigger.Binding = new Binding("mappings[" + i + "].Instrument") { Mode =
```

```
BindingMode.OneWay, UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged };
dataTrigger.Value = null;
dataTrigger.Setters.Add(new Setter(GridViewCell.ToolTipProperty, "Instrument not found in
system"));
cellStyle.Triggers.Add(dataTrigger);

PinMapGrid.Columns.Add(new GridViewComboBoxColumn()
{
UniqueName = "Instrument " + (i + 1),
Header = "Instrument: ",
DataMemberBinding = new Binding("mappings[" + i + "].Instrument"),
DisplayMemberPath = "InstrumentName",
ItemsSource = instrumentSource,
ColumnGroupName = ("Site " + (i + 1)),
MinWidth = 75,
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick,

CellTemplate = cellTemplate,
CellStyle = cellStyle
});
PinMapGrid.Columns.Add(new GridViewComboBoxColumn()
{
UniqueName = "IO " + (i + 1),
Header = "I/O: ",
DataMemberBinding = new Binding("mappings[" + i + "].IO"),
ItemsSourceBinding = new Binding("mappings[" + i + "].Ports"),
ColumnGroupName = ("Site " + (i + 1)),
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick
});
}
}
else //Used for new projects or loading empty pinmaps.
{
//BUG: Adding a site immediately when loading pin map with no pins crashes application.
//Rf pins
PinMapGrid.ColumnGroups.Add(new GridViewColumnGroup() { Name = ("Site " +
MaxMappingsCount), Header = new System.Windows.Controls.TextBlock() { Text = "SITE " +
(MaxMappingsCount + 1), VerticalAlignment = VerticalAlignment.Center, HorizontalAlignment =
HorizontalAlignment.Center } });

string xaml = @"
<DataTemplate xmlns=""http://schemas.microsoft.com/winfx/2006/xaml/presentation"">
```

```
<TextBlock Foreground=""{Binding Path=mappings[" + MaxMappingsCount + @"].Instrument,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged, Converter={StaticResource
NullToColorConverter}}"">
<TextBlock.Text>
<PriorityBinding>
<Binding Path=""mappings[" + MaxMappingsCount + @"].InstrumentName"" Mode=""OneWay""
UpdateSourceTrigger=""PropertyChanged"" IsAsync=""True"" />
<Binding Path=""InstrumentName"" Mode=""OneWay""
UpdateSourceTrigger=""PropertyChanged"" IsAsync=""True"" />
</PriorityBinding>
</TextBlock.Text>
</TextBlock>
</DataTemplate>
";
DataTemplate cellTemplate = (DataTemplate)XamlReader.Parse(xaml);

// Define the CellStyle
Style cellStyle = new Style(typeof(GridViewCell));
DataTrigger dataTrigger = new DataTrigger();
dataTrigger.Binding = new Binding("mappings[" + MaxMappingsCount + "].Instrument") { Mode =
BindingMode.OneWay, UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged };
dataTrigger.Value = null;
dataTrigger.Setters.Add(new Setter(GridViewCell.ToolTipProperty, "Instrument not found in
system"));
cellStyle.Triggers.Add(dataTrigger);

PinMapGrid.Columns.Add(new GridViewComboBoxColumn()
{
UniqueName = "Instrument " + (MaxMappingsCount + 1),
Header = "Instrument: ",
DataMemberBinding = new Binding("mappings[" + MaxMappingsCount + "].Instrument"),
DisplayMemberPath = "InstrumentName",
ItemsSource = instrumentSource,
ColumnGroupName = "Site " + MaxMappingsCount,
Width = 100,
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick,

CellTemplate = cellTemplate,
CellStyle = cellStyle
});
PinMapGrid.Columns.Add(new GridViewComboBoxColumn()
{
```

```
UniqueName = "IO " + (MaxMappingsCount + 1),
Header = "I/O: ",
DataMemberBinding = new Binding("mappings[" + MaxMappingsCount + "].IO"),
ItemsSourceBinding = new Binding("mappings[" + MaxMappingsCount + "].Ports"),
ColumnGroupName = "Site " + MaxMappingsCount,
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick
});
//CreateSiteGUI();  //No Mappings to base site creation off of and there needs to be at least one site to avoid bugs.
}

}
catch(Exception ex ) { MessageBox.Show(ex.Message); }
}

//Generates the collection of strings used as the itemsource for AvailablePinsCollection. (Hard coded / no dynamic pin finding).
private void GenerateAvailablePins()
{
int columnCount = ParentProject.ParameterMapTags.Count;
List<string> RFPins = new List<string>();

ObservableCollection<string> pins = new ObservableCollection<string> { };
var pinsFromParams = ParentProject.ParameterMapTags;
for (int x = 12; x < pinsFromParams.Count; x++)
{
pins.Add(pinsFromParams[x].ColumnName);
}

foreach (Test test in ParentProject.Tests)
{
string fifthParam = test.Parameters[5].Value.ToString();
if (fifthParam != "" && RFPins.Contains(fifthParam) == false)
RFPins.Add(fifthParam);

string sixthParam = test.Parameters[6].Value.ToString();
if (sixthParam != "" && RFPins.Contains(sixthParam) == false)
RFPins.Add(sixthParam);
}

foreach (string item in RFPins) { pins.Add(item); }
```

```
AvailablePinsCollection = pins;
}

#region Constructor
public event Action ChangeOccured;

public override void PaneClose()
{
this.View = null;
}

//public event Action<string> OnChangesMade = (UserControl) => { };

/// <summary>
/// Constructor for the PinMapViewModel.
/// </summary>
public PinMapViewModel(Type contentType, IFileData dataObject, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataObject, parentProject, parentFolder)
{
//OnChangesMade += (UserControl) =>
//{
//    IsSaveRequired = true;
//};

this.CurrentPinMapModel = (PinMapModel)this.DataObject;

//Temp. Workaround for future backwards compatibility purposes
DataFilePath = dataObject.FilePath;

this.LoadData();
}

~PinMapViewModel()
{
CurrentPinMapModel.SiteAdded -= CreateSiteGUI;
CurrentPinMapModel.SiteRemoved -= RemoveSiteGUI;
}

public override void SaveData()
{
base.SaveData();
//string projectFilePath = _viewModelLocator.GetProjectFilePath();
```

```
//
//BinaryFormatter bf = new BinaryFormatter();
////DataManagement.SavePinsData(ref bf, projectFilePath);
//
//XmlSerializer xmlSerializer = new XmlSerializer(typeof(ObservableCollection<Data.Pin>));
//using (StreamWriter writer = new StreamWriter(projectFilePath + @"\PinMapData.xml"))
//{
//    xmlSerializer.Serialize(writer, CurrentPinMapModel.RfPins);
//}
}

public override void LoadData()
{
base.LoadData();
}

public override void Undo()
{
if (DataObject != null)
{
DataObject.Undo();
}

base.Undo();
}
public override void Redo()
{
if (DataObject != null)
{
DataObject.Redo();
}

base.Redo();
}

public override void ValidateData()
{
//Currently only checks the RF Pins Section. Needs to check all.
bool isValid;
this.CurrentPinMapModel.ValidatePinMapData(new Enum[] { PinMapSection.Rf_Pins }, out
isValid);
///TASK: See if all the rules apply to every section of the pin map or if each section has it's own set
```

```
of rules !!!

base.ValidateData();
}

public override void RelinkData()
{
foreach (var pin in this.CurrentPinMapModel.RfPins)
{
foreach(var mapping in pin.mappings)
{
foreach (var instrument in this.MVM.MySystem.RF_Instruments.Where(i => i.InstrumentName ==
mapping.InstrumentName))
{
mapping.Instrument = instrument;
}
}
}
foreach (var pin in this.CurrentPinMapModel.DigitalPins)
{
foreach (var mapping in pin.mappings)
{
foreach (var instrument in this.MVM.MySystem.PXI_Instruments.Where(i => i.InstrumentName ==
mapping.InstrumentName))
{
mapping.Instrument = instrument;
}
}
}
foreach (var pin in this.CurrentPinMapModel.PowerSupplyPins)
{
foreach (var mapping in pin.mappings)
{
foreach (var instrument in this.MVM.MySystem.Power_Instruments.Where(i => i.InstrumentName
== mapping.InstrumentName))
{
mapping.Instrument = instrument;
}
}
}

base.RelinkData();
```

```
}


private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
this.ValidateData();
}
#endregion

#region Commands

public ICommand ClearRowSelectionCommand { get { return new
DelegateCommand(ClearRowSelection); } }
private void ClearRowSelection(object obj)
{
View.UnselectAll();
}

public ICommand AddPinCommand => new RelayCommand(AddPin);
private void AddPin()
{
PinMapSection CurrentSection =
PinMapModel.GetPinMapSectionEnum(SelectedPinMapSection);
CurrentPinMapModel.AddPin(CurrentSection, null);

OnChangeOccured();
}

public ICommand AddSiteCommand => new RelayCommand(AddSite);
private void AddSite()
{
CurrentPinMapModel.AddSite();

OnChangeOccured();
}

public ICommand RemovePinCommand => new RelayCommand(RemovePin);
private void RemovePin()
{
if (PinMapGrid.SelectedItem != null)
```

```
{
Data.PinModel SelectedPin = PinMapGrid.SelectedItem as Data.PinModel;
PinMapSection section = PinMapModel.GetPinMapSectionEnum(SelectedPinMapSection);

CurrentPinMapModel.RemovePin(section, SelectedPin);
}

OnChangeOccured();
}


//Opens and Prepares Remove site window.
public ICommand ShowRemoveSiteWindowCommand => new
RelayCommand(ShowRemoveSiteWindow);
private void ShowRemoveSiteWindow()
{
RadWindow RemoveSiteWindow = new RadWindow()
{
Width = 400,
Height = 250,
WindowStartupLocation = System.Windows.WindowStartupLocation.CenterScreen,
Content = new UserControls.RemoveSiteWindow() { DataContext = this },
Header = "Remove Site"
};
RemoveSiteWindow.Show(); //Opens active sites window for deletion of desired site.
}

//Fires when hit done from removeSiteWindow.
public ICommand SiteRemovalCommand => new DelegateCommand(SiteRemoval);
private void SiteRemoval(object obj)
{
if(obj != null)
{
Site siteToRemove = (obj as Site);
if(siteToRemove.SiteID == 1)
{
Console.WriteLine("Removal of 'Site 1' is currently not allowed.");
}
else
{
CurrentPinMapModel.RemoveSite(siteToRemove);
```

```
OnChangeOccured();
}
}
}

private void CreateSiteGUI()
{
//Used to create the GUI columns and column groups with the dynamic binding.
try
{
//ObservableCollection<Data.Instrument> source = null;
//switch (PinMapModel.GetPinMapSectionEnum(SelectedPinMapSection.Content.ToString()))
//{
//    case PinMapSection.Rf_Pins:
//        source = MVM.MySystem.RF_Instruments;
//        break;
//    case PinMapSection.Digital_Pins:
//        source = MVM.MySystem.PXI_Instruments;
//        break;
//    case PinMapSection.Power_Supply_Pins:
//        source = MVM.MySystem.Power_Instruments;
//        break;
//}
//
////BUG: Adding a site immediately when loading pin map with no pins crashes application.
////Rf pins
//PinMapGrid.ColumnGroups.Add(new GridViewColumnGroup() { Name = ("Site " +
MaxMappingsCount), Header = ("Site " + (MaxMappingsCount + 1)) });
//PinMapGrid.Columns.Add(new GridViewComboBoxColumn()
//{
//    UniqueName = "InstrumentName " + (MaxMappingsCount + 1),
//    Header = "Instrument: ",
//    DataMemberBinding = new Binding("mappings[" + MaxMappingsCount + "].InstrumentName"),
//    SelectedValueMemberPath = "InstrumentName",
//    DisplayMemberPath = "InstrumentName",
//    //ItemsSourceBinding = new Binding("mappings[" + i + "].Instruments"), (( ATTN: If used you
will need to place an instruments collection back into the maping object.))
//    ItemsSource = source,
//    ColumnGroupName = "Site " + MaxMappingsCount,
//    Width = 100,
//    EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick
//});
```

```
//PinMapGrid.Columns.Add(new GridViewComboBoxColumn()
//{
//    UniqueName = "Ports " + (MaxMappingsCount + 1),
//    Header = "I/O: ",
//    DataMemberBinding = new Binding("mappings[" + MaxMappingsCount + "].IO"),
//    ItemsSourceBinding = new Binding("mappings[" + MaxMappingsCount + "].Ports"),
//    ColumnGroupName = "Site " + MaxMappingsCount,
//    EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick
//});
PinMapGrid.ColumnGroups.Clear();
PinMapGrid.Columns.Clear();
RebindPinMapGridView();

OnChangeOccured();
}
catch (Exception Ex)
{
System.Windows.MessageBox.Show(Ex.Message + " || Add Pin(s) before adding and additional
sites.");
}
}
private void RemoveSiteGUI()
{
//Clean up after removal of site mappings.
SelectedSite = null;
PinMapGrid.ColumnGroups.Clear();
PinMapGrid.Columns.Clear();
RebindPinMapGridView(); // Needs a rebind incase a middle site is chosen for deletion.

OnChangeOccured();
}

//Code needs to be updated with the current pin map sections code base.
#region ImportPins Window Command Functions

private void ResetImportPins()
{
ImportPinsCollection.Clear();
AvailablePinsCollection.Clear();
SelectedImportPin.Clear();
SelectedAvailablePin.Clear();
}
```

```
public ICommand ImportPinsCommand => new RelayCommand(ImportPins);
private void ImportPins()
{
GenerateAvailablePins();
ImportPinWindow.DataContext = this;
ImportPinWindow.Show();
}

public ICommand CancelBtnCommand => new RelayCommand(CancelClicked);
private void CancelClicked()
{
ImportPinWindow.Close();
ResetImportPins();
}

//NEEDS TO BE UPDATED WITH CURRENT PIN MAP SECTIONS.
public ICommand DoneBtnCommand => new RelayCommand(DoneClicked);
private void DoneClicked()
{
if(ImportPinsCollection != null)
{
foreach (string S in ImportPinsCollection)
{
//This code does not accurately import pins into the correct PinMap section.
PinMapSection CurrentSection =
PinMapModel.GetPinMapSectionEnum(SelectedPinMapSection);
CurrentPinMapModel.AddPin(CurrentSection, S);

//old code...
//Data.Pin P = new Data.Pin() { PinName = S };
//AddGeneralPin(P); //Was used to generate the sites of the pin.
}
//CreateSite();
//PinMapGrid.Rebind();
}

ImportPinWindow.Close();
ResetImportPins();

OnChangeOccured();
}
```

```
public ICommand MovePinCommand => new DelegateCommand(MovePin);
private void MovePin(object obj)
{
string ToLocation = obj.ToString();
switch (ToLocation)
{
case "ToImportPins":
foreach(string IP in SelectedAvailablePin)
{
AvailablePinsCollection.Remove(IP);
ImportPinsCollection.Add(IP);
}
break;
case "ToAvailablePins":
foreach(string AP in SelectedImportPin)
{
ImportPinsCollection.Remove(AP);
AvailablePinsCollection.Add(AP);
}
break;
default:
break;
}
}

public ICommand SelectAllAvailablePinsCommand => new
RelayCommand(SelectAllAvailablePins);
private void SelectAllAvailablePins()
{
if(AvailablePinsCollection != null)
{
foreach (string IP in AvailablePinsCollection)
{
SelectedAvailablePin.Add(IP);
}
}
}

#endregion
```

```
#endregion

#region Helper Methods

public void OnClosed(object sender, WindowClosedEventArgs e)
{
UserDialogEntry = null;
if (e.PromptResult != null)
{
UserDialogEntry = e.PromptResult.ToString();
}
}

#endregion
}
}
```

AddNewFileDialogVM.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.Data.Models.TestModels;
using MerlinTestStudio_Demo_Telerik.UserControls.Patterns;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.DialogViewModels
{
public class AddNewFileDialogVM : ViewModelBase
{
#region Private Members
private NewItem _selectedItem = null;
private ObservableCollection<NewItem> _newItemsCollection = new
ObservableCollection<NewItem>();
```

```
#endregion

//GUI
public NewItem SelectedItem
{
get { return _selectedItem; }
set
{
_selectedItem = value;
OnPropertyChanged("SelectedItem");
if(value != null)
{
this.CustomFileName = value.ItemName;
}
}
}
public ObservableCollection<NewItem> NewItemsCollection
{
get { return _newItemsCollection; }
set { _newItemsCollection = value; OnPropertyChanged("NewItemsCollection"); }
}

private string _customFileName = string.Empty;
public string CustomFileName
{
get { return _customFileName; }
set { _customFileName = value; OnPropertyChanged("CustomFileName"); }
}

//File Production
public string DataFilePathProduced { get; set; }
public IFileData FileObjProduced { get; set; } //Soon to be old.
private ProjectFolder FolderObj { get; set; }

public AddNewFileDialogVM(ProjectFolder folderObj)
{
this.FolderObj = folderObj;
switch (folderObj.Section) //Create new items source based on the Project file section this window was opened from.
{
case ProjectFileSection.Product_Configurations:
this.NewItemsCollection = new ObservableCollection<NewItem>()
```

```
{
new NewItem() { ItemName =  "Product Configuration", Extension =
BackendConstants.MerlinProduct_Config_Extension, Description = $"Product Configuration File
({BackendConstants.MerlinProduct_Config_Extension}) Description."},
};
break;
case ProjectFileSection.Test_Limits:
this.NewItemsCollection = new ObservableCollection<NewItem>()
{
new NewItem() { ItemName =  "Test Limits", Extension = ".limits", Description = "Test Limits File
(.limits) Description."},
new NewItem() { ItemName =  "Limit Golds", Extension = ".golds",Description = "Limit Golds File
(.golds) Description."},
new NewItem() { ItemName =  "Limit Offsets", Extension = ".offsets",Description = "Limit Offsets
(.offfsets) Description."},
new NewItem() { ItemName =  "Limit Trace Loss", Extension = ".traceloss",Description = "Limit
Trace Loss File (.traceloss) Description."},
new NewItem() { ItemName =  "Test Fixtures", Extension = ".fixtures",Description = "Test Fixtures
File (.fixtures) Description."}
};
break;
case ProjectFileSection.Test_Parameters:
break;
case ProjectFileSection.Calibration_Definitions:
this.NewItemsCollection = new ObservableCollection<NewItem>()
{
new NewItem() { ItemName =  "Calibration Definition", Extension =
BackendConstants.MTS_Cal_Config_Extension, Description = $"Calibration Definition File
({BackendConstants.MTS_Cal_Config_Extension}) Description."},
};
break;
case ProjectFileSection.Connection_Diagrams:
this.NewItemsCollection = new ObservableCollection<NewItem>()
{
new NewItem() { ItemName =  "Connection Diagram", Extension = ".xml", Description =
"Connection Diagram File (.xml) Description."},
};
break;
case ProjectFileSection.Digital_Patterns:
this.NewItemsCollection = new ObservableCollection<NewItem>()
{
new NewItem() { ItemName =  "Digital Levels", Extension = ".digilevels", Description = "Digital
```

```
Levels File (.digilevels) Description."},
new NewItem() { ItemName =  "Digital Timing", Extension = ".digitiming",Description = "Digital
Timing File (.digitiming) Description."},
new NewItem() { ItemName =  "RFFE Digital Pattern", Extension = ".rffepat",Description = "Digital
Pattern File (.rffepat) Description."},
new NewItem() {ItemName = "Generic RFFE Digital Pattern", Extension = ".genericrffepattern",
Description = "Generic RFFE Digital Pattern File (.genericrffepattern) Description."}
};
break;
}
}

public ICommand AddCommand { get { return new DelegateCommand(AddNewItem); } }
private void AddNewItem(object obj)
{
if (SelectedItem == null)
{
return;
}
if (string.IsNullOrEmpty(this.CustomFileName))
{
return;
}

switch (FolderObj.Section)
{
case ProjectFileSection.Product_Configurations:
string pcUniqueFileName = MainWindowViewModel.GetUniqueFileName(FolderObj,
this.CustomFileName);
string pcFilePath =
Path.Combine(FolderObj.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Product
_Configurations), $"{pcUniqueFileName}{SelectedItem.Extension}");
switch (SelectedItem.Extension)
{
case BackendConstants.MerlinProduct_Config_Extension:
DataFilePathProduced = pcFilePath;
break;
}
break;
case ProjectFileSection.Test_Limits:
string limitUniqueFileName = MainWindowViewModel.GetUniqueFileName(FolderObj,
this.CustomFileName);
```

```
string limitFilePath =
Path.Combine(FolderObj.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Test_Limits), $"{limitUniqueFileName}{SelectedItem.Extension}");
switch (SelectedItem.Extension)
{
case ".limits":
DataFilePathProduced = limitFilePath;
//FileObjProduced = new TestLimitsModel(limitFilePath);
break;
case ".golds":
DataFilePathProduced = limitFilePath;
break;
case ".offsets":
DataFilePathProduced = limitFilePath;
break;
case ".traceloss":
DataFilePathProduced = limitFilePath;
break;
case ".fixtures":
DataFilePathProduced = limitFilePath;
break;
}
break;
case ProjectFileSection.Calibration_Definitions:
string cdmUniqueFileName = MainWindowViewModel.GetUniqueFileName(FolderObj,
this.CustomFileName);
string cdmFilePath =
Path.Combine(FolderObj.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Project_Data), "Cal Data", $"{cdmUniqueFileName}{SelectedItem.Extension}");
switch (SelectedItem.Extension)
{
case BackendConstants.MTS_Cal_Config_Extension:
DataFilePathProduced = cdmFilePath;
break;
}
break;
case ProjectFileSection.Connection_Diagrams:
string cdUniqueFileName = MainWindowViewModel.GetUniqueFileName(FolderObj,
this.CustomFileName);
string cdFilePath =
Path.Combine(FolderObj.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Connection_Diagrams), $"{cdUniqueFileName}{SelectedItem.Extension}");
```

```
switch (SelectedItem.Extension)
{
case ".xml":
DataFilePathProduced = cdFilePath;
break;
}
break;
case ProjectFileSection.Digital_Patterns:
string dpuniqueFileName = MainWindowViewModel.GetUniqueFileName(FolderObj,
this.CustomFileName);
string dpfilePath =
Path.Combine(FolderObj.ParentProject.GetProjectFileSectionDirectory(ProjectFileSection.Digital_
Patterns), $"{dpuniqueFileName}{SelectedItem.Extension}");
switch (SelectedItem.Extension)
{
case ".genericrffepattern":
DataFilePathProduced = dpfilePath;
break;
case ".rffepat":
DataFilePathProduced = dpfilePath;
break;
case ".digilevels":
DataFilePathProduced = dpfilePath;
break;
case ".digitiming":
DataFilePathProduced = dpfilePath;
break;
}
break;
}

}
}

public class NewItem
{
public string ImageData { get { return @"\Resources\Images\TreeItems\Document_16x.png"; } set
{ } }
public string ItemName { get; set; }
public string Extension { get; set; }
public string Description { get; set; }
}
```

}

CloseProjectSaveDiaglogViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ComponentModel;
using System.Windows;
using MerlinTest.Tools.TestStudio.Model;
using System.Collections.ObjectModel;
using System.Windows.Media.Imaging;
using System.Windows.Input;
using System.IO;
using MT.TestStudio.Exceptions;
using System.Windows.Controls;
using MerlinTestStudio_Demo_Telerik.ViewModels;

namespace MT.TestStudio.GUI.ViewModels
{
/// <summary>
/// ViewModel for new project UserControl.
/// </summary>
public class CloseProjectSaveDiaglogViewModel : UcViewModelBase, INotifyPropertyChanged
{
#region Private Member Variables

/// <summary>
/// Gets / Sets  reference to the MainWindowViewModel.
/// </summary>
private MainWindowViewModel mwvm { get; set; }

/// <summary>
/// Gets / Sets the windows style.
/// </summary>
private WindowStyled hostWindow { get; set; }

#endregion Private Member Variables

#region Constructor
```

```
/// <summary>
/// Constructor for the NewProjectDiaglogViewModel.
/// </summary>
public CloseProjectSaveDiaglogViewModel(MainWindowViewModel MainWindowViewModel,
WindowStyled HostWindow)
{
// Don't execute the viewmodel code if we're just looking at the XAML in the designer.
if (DesignerProperties.GetIsInDesignMode(new DependencyObject())) return;

mwvm = MainWindowViewModel;

hostWindow = HostWindow;
}

#endregion Constructor

#region Public Properties

/// <summary>
/// Observable list of projects.
/// </summary>
public ObservableCollection<string> FileToSaveList { get; set; } = new
ObservableCollection<string>();

/// <summary>
/// List of viewModels that require to be saved.
/// </summary>
public List<PVM> ObjectsToSave { get; set; } = new List<PVM>();

#endregion Public Properties

#region Public Methods

/// <summary>
/// Method that gets called whenever the Model has updated a value.
/// </summary>
/// <param name="sender"></param>
/// <param name="e">Event argument that contains the property name that has
changed.</param>
private void ModelObj_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
if (e.PropertyName == "CsvDataLoaded")
```

```
{
// Not implemented yet.
}
}

#endregion Public Methods

#region Commands

#region DialogButton

/// <summary>
/// Diaglog Button
/// </summary>
public ICommand DialogButton
{
get
{
return new RelayCommand<object>(DialogButtonCommandExecute,
CanDialogButtonCommandExecute);
}
}

/// <summary>
///  Diaglog Button
/// </summary>
private void DialogButtonCommandExecute(object parameter)
{
try
{
string commandParameter = parameter as string;

if (commandParameter != null)
{
switch (commandParameter)
{
case "Yes":
{
// Work out which files need to be saved.
foreach (PVM pvm in this.ObjectsToSave)
{
if (pvm.IsSaveRequired == true)
```

```
{
pvm.SaveData();
}
}

// Indicate the diaglog exited by user consent ( not cancel or F4 )
hostWindow.DialogResult = true;
break;
}
case "No":
{
// Indicate the diaglog exited by user consent ( not cancel or F4 )
hostWindow.DialogResult = true;
break;
}
default:
{
this.mwvm.trace.TraceWarning("Unknown value of commandParameter {0}.(defaulting to
cancel(NULL)", commandParameter);
hostWindow.DialogResult = null;
break;
}
}

}

hostWindow.Close();

}
catch (Exception ex)
{
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}
}

/// <summary>
/// Can the command execute.
/// </summary>
/// <returns>True (always)</returns>
private bool CanDialogButtonCommandExecute(object parameter)
{
return true;
```

```
}

#endregion DialogButton

#endregion Commands
}
}
```

NewProjectDiaglogViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ComponentModel;
using System.Windows;
using MerlinTest.Tools.TestStudio.Model;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using System.Collections.ObjectModel;
using System.Windows.Media.Imaging;
using System.Windows.Input;
using System.IO;
using MT.TestStudio.Exceptions;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using MerlinTestStudio_Demo_Telerik.Data.Services;
using MerlinTestStudio_Demo_Telerik.Data;

namespace MT.TestStudio.GUI.ViewModels
{
public enum ProjectType
{
Empty,
Default,
Clone
}

public class ProjectInfo
{
public string ProjectName { get; set; }

public string ProjectDescription { get; set; }
```

```
public BitmapImage ImageData {get; set; }

public ProjectType ProjectType { get; set; }

public bool IsProjectEnabled { get; set; }
}

/// <summary>
/// ViewModel for new project UserControl.
/// </summary>
public class NewProjectDiaglogViewModel : UcViewModelBase, INotifyPropertyChanged
{
#region Private Member Variables

/// <summary>
/// Instance of the Model Object.
/// </summary>
// private PinMapModel ModelObj { get; set; }

/// <summary>
/// Gets / Sets  reference to the MainWindowViewModel.
/// </summary>
private MainWindowViewModel mwvm { get; set; }

/// <summary>
/// Gets / Sets the windows style.
/// </summary>
private WindowStyled hostWindow { get; set; }

/// <summary>
/// Gets / Sets the currently selected project.
/// </summary>
private ProjectInfo selectedProject;

private string _targetSystem = "APS-500";

private string _testProgram = string.Empty;

/// <summary>
/// Gets / Sets the file path of the project location.
/// </summary>
private string location;
```

```
/// <summary>
/// Gets / Sets the error message to be displayed.
/// </summary>
private string errorMessage;

/// <summary>
/// Gets / Sets project name.
/// </summary>
private string projectName;

/// <summary>
/// Gets / Sets the product name.
/// </summary>
private string productName;

/// <summary>
/// Gets / Sets if the Diaglog OK button should be enabled.
/// </summary>
private bool okButtonEnabled;

#endregion Private Member Variables

#region Constructor
//IProjectConfigurationService _projectConfigurationService;

/// <summary>
/// Constructor for the NewProjectDiaglogViewModel.
/// </summary>
public NewProjectDiaglogViewModel(MainWindowViewModel MainWindowViewModel,
WindowStyled HostWindow)
{
//_projectConfigurationService = projectConfigurationService;

// Don't execute the viewmodel code if we're just looking at the XAML in the designer.
if (DesignerProperties.GetIsInDesignMode(new DependencyObject())) return;

mwvm = MainWindowViewModel;

hostWindow = HostWindow;

// ModelObj = new PinMapModel();
```

```
// Set some defaults.
this.ProjectsList.Add(new ProjectInfo() { ProjectName = "Merlin Test Empty Project", ProjectType = ProjectType.Empty, ProjectDescription = "A standard Merlin Test project with no files but all the project folders created.", ImageData = LoadImage("..\\Images\\NewDocument_32x32.png"), IsProjectEnabled = true });
this.ProjectsList.Add(new ProjectInfo() { ProjectName = "Merlin Test Default Project Files", ProjectType = ProjectType.Default, ProjectDescription = "A standard Merlin Test project with default files added.", ImageData = LoadImage("..\\Images\\PrintEntireDocument.png"), IsProjectEnabled = false });

this.SelectedProject = this.ProjectsList.FirstOrDefault();

this.ProjectName = "New Project"; //Also sets the ProductName.
//this.ProductName = "Product001";
this.Location = BackendConstants.InitialProjectDirectory;
}

#endregion Constructor

#region Public Properties
public IEnumerable<string> TargetSystemOptions { get { return BackendConstants.TargetSystemOptions; } }
public string UserTargetSystem
{
get { return _targetSystem; }
set { _targetSystem = value; OnPropertyChanged("UserTargetSystem"); }
}
public string TestProgram
{
get { return _testProgram; }
set { _testProgram = value; OnPropertyChanged("TestProgram"); }
}

/// <summary>
/// Gets / Sets the error message to be displayed.
/// </summary>
public string ErrorMessage
{
get
{
return errorMessage;
```

```
}
set
{
errorMessage = value;
OnPropertyChanged("ErrorMessage");
}
}

/// <summary>
/// Gets / Sets the file path of the project location.
/// </summary>
public string Location
{
get
{
return location;
}
set
{
location = value;
OnPropertyChanged("Location");
}
}

/// <summary>
/// Gets / Sets project name.
/// </summary>
public string ProjectName
{
get
{
return projectName;
}
set
{
projectName = value;
ProductName = value;
OnPropertyChanged("ProjectName");
}
}

/// <summary>
```

```
/// Gets / Sets product name.
/// </summary>
public string ProductName
{
get
{
return productName;
}
set
{
productName = value;
OnPropertyChanged("ProductName");
}
}

/// <summary>
/// Gets / Sets the currently selected project.
/// </summary>
public ProjectInfo SelectedProject
{
get
{
return selectedProject;
}
set
{
selectedProject = value;

OnPropertyChanged("SelectedProject");
}
}

/// <summary>
/// Observable list of projects.
/// </summary>
public ObservableCollection<ProjectInfo> ProjectsList { get; set; } = new
ObservableCollection<ProjectInfo>();

/// <summary>
/// Gets / Sets if the OK button should be enabled.
/// </summary>
public bool OkButtonEnabled
```

```
{
get
{
return okButtonEnabled;
}
set
{
okButtonEnabled = value;
OnPropertyChanged("OkButtonEnabled");
}
}


#endregion Public Properties

#region Public Methods

/// <summary>
/// Method that gets called whenever the Model has updated a value.
/// </summary>
/// <param name="sender"></param>
/// <param name="e">Event argument that contains the property name that has
changed.</param>
private void ModelObj_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
if (e.PropertyName == "CsvDataLoaded")
{
// Not implemented yet.
}
}

#endregion Public Methods

#region Commands

#region Create Project

/// <summary>
/// Create new Project
/// </summary>
public ICommand CreateProjectCommand
{
```

```
get
{
return new RelayCommand(CreateProjectCommandExecute,
CanCreateProjectCommandExecute);
}
}

/// <summary>
/// Create new project.
/// </summary>
private void CreateProjectCommandExecute()
{
try
{
this.ErrorMessage = string.Empty;

var combinedPathAndProjectFile = Path.Combine(this.Location, this.ProjectName,
this.ProjectName + BackendConstants.MerlinProjectExtension);
//var combinedPathAndProjectFile = Path.Combine(this.Location, this.ProjectName,
this.ProjectName + BackendConstants.MerlinSolutionExtension);

switch (this.SelectedProject.ProjectType)
{
case ProjectType.Empty:
{
#region New Mutli Project Only Code Implementation
//Create top level product/project folder to house ethe .mtproj file and its contents.
MerlinProject projectBase = new MerlinProject(Path.Combine(this.Location, this.ProjectName))
{
ProductName = this.ProductName,
TestProgram = this.TestProgram,
UserTargetSystem = this.UserTargetSystem
};
projectBase.CreateProjectFileStructure();
this.mwvm.ProjectsCollection.Add(projectBase);
//Log as opened recently.
this.mwvm.MyRecentData.AddRecentProject(projectBase);
//this.mwvm.LoadLayout();

///Save then reload on project creation to make ensure both methods work and that no progress is
lost on relaod of the same project in a different instance of MTS.
```

```
#endregion
#region Soltuion .mtsln code
//MerlinSolution newSolution = new MerlinSolution(Path.Combine(this.Location,
this.SolutionName), this.mwvm);
//MerlinProject projectBase = new MerlinProject(Path.Combine(this.Location, this.SolutionName,
this.ProjectName), newSolution)
//{
//    ProductName = this.ProductName,
//    TestProgram = this.TestProgram,
//    UserTargetSystem = this.UserTargetSystem
//};
////this.mwvm.CurrentSolution.Projects.Add(projectBase);
//newSolution.CreateSoltutionFileStructure(projectBase);
//this.mwvm.CurrentSolution = newSolution;
//this.mwvm.LoadLayout();
#endregion
#region Old .mtp project code
//// Create an new empty project.
//this.mwvm.ProjectFileIoObj.CreateNewEmptyProject(this.Location, this.ProjectName,
this.ProductName, this.UserTargetSystem, this.TestProgram);
//
//// Now the project has been created load the project.
//this.mwvm.LoadProjectTree(combinedPathAndProjectFile);
#endregion
break;
}
case ProjectType.Default:
{
//// Create an new empty project.
//this.mwvm.ProjectFileIoObj.CreateNewEmptyProject(this.Location, this.ProjectName,
this.ProductName, this.UserTargetSystem, this.TestProgram);
//
//// Now the project has been created load the project.
//this.mwvm.LoadProjectTree(combinedPathAndProjectFile);
//
//this.mwvm.AddFileCommandExecute("Product Configuration");
////this.mwvm.AddFileCommandExecute("Test Limits");
//this.mwvm.AddFileCommandExecute("Cal Def V5");

break;
}
}
```

```
hostWindow.DialogResult = true;

this.mwvm.trace.TraceInformation("File " + combinedPathAndProjectFile + " succesfully created.");

this.mwvm.Status = "File " + combinedPathAndProjectFile + " succesfully created.";

hostWindow.Close();

}
catch(ProjectFileIOException ex)
{
this.ErrorMessage = ex.Message;
}
catch (Exception ex)
{
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}
}

/// <summary>
/// Can the command execute.
/// </summary>
/// <returns>True (always)</returns>
private bool CanCreateProjectCommandExecute()
{
return true;
}

#endregion Create Project

#region Browse Folder

/// <summary>
/// Opens a dialog to set the folder to create the project in.
/// </summary>
public ICommand ProjectFolderCommand
{
get
{
return new RelayCommand(ProjectFolderCommandExecute,
CanProjectFolderCommandExecute);
```

```
}
}

/// <summary>
/// Opens a dialog to set the folder to create the project in.
/// </summary>
private void ProjectFolderCommandExecute()
{
try
{
// WPF doesn't have a FolderBrowser control so we can use the windows forms one.
var dlg = new System.Windows.Forms.FolderBrowserDialog();

dlg.SelectedPath = this.Location;

// Turn off the Create New Folder Button.
dlg.ShowNewFolderButton = false;

// Add a description to the dialog box.
dlg.Description = "Select a path that the Data Logger Viewer will use to search for valid results files.";

// Show the Diaglog.
dlg.ShowDialog();

// Now update the currently selected item in the lists Path value with the new one the user just added.
this.Location = dlg.SelectedPath;
}
catch (Exception ex)
{
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}
}

/// <summary>
/// Can the command execute.
/// </summary>
/// <returns>True (always)</returns>
private bool CanProjectFolderCommandExecute()
{
return true;
```

```
}

#endregion Browse Folder

#endregion Commands

#region Private Methods

// for this code image needs to be a project resource
private BitmapImage LoadImage(string filename)
{
return new BitmapImage(new Uri("pack://application:,,,/" + filename));
}

#endregion Private Methods
}
}
```

DiagramViewModel.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Windows;
using System.Xml.Serialization;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows.Controls.Diagrams.Extensions.ViewModels;
using Telerik.Windows.Diagrams.Core;

namespace MerlinTestStudio_Demo_Telerik.GraphViewModels
{
public interface IDiagramView
{
//GraphSource GetGraphSource();

//void SetGraphSource(GraphSource graphSourceToSet);
```

```
//Saves the diagram to a string so we can write it to an XML file.
string GetDiagramSerString();

//Loads the diagram by setting a Deserialization string.
void SetDiagramDeSerString(string DeSerString);
}

public class DiagramViewModel: PVM
{
private IDiagramView _view;
private string _diagramObj = string.Empty;
private object _selectedDiagramElement;

#region Public

public IDiagramView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");
if (value != null)
{
View.SetDiagramDeSerString(DiagramObj);
//View.RebindTestLimitsGridView();
//this.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}

public string DiagramObj
{
get
{
return _diagramObj;
}
set
{
_diagramObj = value;
OnPropertyChanged("DiagramObj");
}
```

```
}

public object SelectedDiagramElement
{
get { return _selectedDiagramElement; }
set
{
if (value != null)
{
_selectedDiagramElement = value;
OnPropertyChanged("SelectedDiagramElement");
}
}
}
#endregion

#region Constructor
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;

public override void PaneClose()
{
//this.CommitEditBeforeSave -= View.ForceCommitEdit;
//this.View.Dispose();
DiagramObj = View.GetDiagramSerString();
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
}
}

public DiagramViewModel(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
```

```
{
this.DataFilePath = dataFilePath;

this.LoadData();

this.ChangeOccured += ChangeOccuredSubscriber;
}

public override void SaveData()
{
CommitEditBeforeSave?.Invoke();

//this.DiagramObj.SaveToXml(this.DataFilePath);
try
{
//XmlSerializer xmlSerializer = new XmlSerializer(typeof(RadDiagram));

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(this.DataFilePath))
{
writer.Write(View != null ? View.GetDiagramSerString() : this.DiagramObj);
//xmlSerializer.Serialize(writer, this.DiagramObj);
}
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public override void LoadData()
{
string dataModel = null;

if (File.Exists(this.DataFilePath))
{
try
{
//XmlSerializer ser = new XmlSerializer(typeof(RadDiagram));

// Deserialize XML file.
using (StreamReader sr = new StreamReader(this.DataFilePath))
```

```
{
dataModel = sr.ReadToEnd();
//dataModel = (RadDiagram)ser.Deserialize(sr);
}
}
catch (Exception ex) { System.Windows.MessageBox.Show(ex.Message); }
}

if (dataModel != null)
{
this.DiagramObj = dataModel;
//this.ParentProject.TestLimits.Add(dataModel); //For IFileData Collections.
}

base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{

}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
#endregion

#region Commands
//Commands here.
#endregion
}

//public class DiagramViewModel : ObservableGraphSourceBase<NodeViewModelBase,
LinkViewModelBase<NodeViewModelBase>>
//{
//    public DiagramViewModel()
//    {
//        var first = new NodeViewModelBase
//        {
//            Content = "First item",
```

```
//            Position = new Point(50, 100)
//        };
//        var second = new NodeViewModelBase
//        {
//            Content = "Second item",
//            Position = new Point(150, 100),
//            RotationAngle = 45
//        };
//        var third = new NodeViewModelBase
//        {
//            Content = "Third item",
//            Position = new Point(250, 100)
//        };
//        this.AddNode(first);
//        this.AddNode(second);
//        this.AddNode(third);
//        this.AddLink(new LinkViewModelBase<NodeViewModelBase>(first, second));
//        this.AddLink(new LinkViewModelBase<NodeViewModelBase>(second, third));
//    }
//}
[Serializable]
public class CustomConnectorCollection : List<RadDiagramConnector>
{

}
[Serializable]
public class OurInstrument : ContainerNodeViewModelBase<object>
{
//public string Content { get; set; }
}
[Serializable]
public class Port : NodeViewModelBase
{
//public string Content { get; set; }
public bool UseSecondConnectors { get; set; }
public Port()
{

}

}
[Serializable]
```

```
public class Attenuator : NodeViewModelBase
{
//public string Content { get; set; }
}

[Serializable]
public class Link : LinkViewModelBase<NodeViewModelBase>
{
public Link()
{
this.SourceConnectionName = ConnectorPosition.Auto;
this.TargetConnectionName = ConnectorPosition.Auto;
}

private string _sourceConnectionName;
public string SourceConnectionName
{
get { return _sourceConnectionName; }
set
{
_sourceConnectionName = value;
OnPropertyChanged("SourceConnectionName");
}
}

private string _targetConnectionName;
public string TargetConnectionName
{
get { return _targetConnectionName; }
set
{
_targetConnectionName = value;
OnPropertyChanged("TargetConnectionName");
}
}
}

[Serializable]
public class GraphSource : SerializableGraphSourceBase<NodeViewModelBase, Link>
{
//Constructor that sets up the default Items.
public GraphSource()
```

```
{
OurInstrument Device = new OurInstrument() { Content = "Device", Position = new Point(250, 100)
};
Port Site1 = new Port() { Content = "Site 1", Position = new Point(250, 100),
UseSecondConnectors = true };
Port Site2 = new Port() { Content = "Site 2", Position = new Point(475, 100),
UseSecondConnectors = true };
Port Site3 = new Port() { Content = "Site 3", Position = new Point(250, 150),
UseSecondConnectors = true };
Port Site4 = new Port() { Content = "Site 4", Position = new Point(475, 150),
UseSecondConnectors = true };
Port Site5 = new Port() { Content = "Site 5", Position = new Point(250, 200),
UseSecondConnectors = true };
Port Site6 = new Port() { Content = "Site 6", Position = new Point(475, 200),
UseSecondConnectors = true };
Device.AddItem(Site1);
Device.AddItem(Site2);
Device.AddItem(Site3);
Device.AddItem(Site4);
Device.AddItem(Site5);
Device.AddItem(Site6);

OurInstrument RF100 = new OurInstrument() { Content = "RF100", Position = new Point(220, 400)
};
Port P1 = new Port() { Content = "P1", Position = new Point(200, 400), UseSecondConnectors =
true };
Port P2 = new Port() { Content = "P2", Position = new Point(200, 450), UseSecondConnectors =
true };
Port P3 = new Port() { Content = "P3", Position = new Point(200, 500), UseSecondConnectors =
true };
Port P4 = new Port() { Content = "P4", Position = new Point(200, 550), UseSecondConnectors =
true };
Port P5 = new Port() { Content = "P5", Position = new Point(200, 600), UseSecondConnectors =
true };
Port P6 = new Port() { Content = "P6", Position = new Point(200, 650), UseSecondConnectors =
true };
Port P7 = new Port() { Content = "P7", Position = new Point(200, 700), UseSecondConnectors =
true };
Port P8 = new Port() { Content = "P8", Position = new Point(200, 750), UseSecondConnectors =
true };
Port P9 = new Port() { Content = "P9", Position = new Point(200, 800), UseSecondConnectors =
true };
```

```
Port P10 = new Port() { Content = "P10", Position = new Point(200, 850), UseSecondConnectors = true };
RF100.AddItem(P1);
RF100.AddItem(P2);
RF100.AddItem(P3);
RF100.AddItem(P4);
RF100.AddItem(P5);
RF100.AddItem(P6);
RF100.AddItem(P7);
RF100.AddItem(P8);
RF100.AddItem(P9);
RF100.AddItem(P10);
Link P1ToSite1 = new Link() { Source = P1, Target = Site1 };

OurInstrument RF200 = new OurInstrument() { Content = "RF200", Position = new Point(520, 400) };
Port M1 = new Port() { Content = "M1", Position = new Point(500, 400), UseSecondConnectors = true };
Port M2 = new Port() { Content = "M2", Position = new Point(500, 450), UseSecondConnectors = true };
Port M3 = new Port() { Content = "M3", Position = new Point(500, 500), UseSecondConnectors = true };
Port M4 = new Port() { Content = "M4", Position = new Point(500, 550), UseSecondConnectors = true };
Port M5 = new Port() { Content = "M5", Position = new Point(500, 600), UseSecondConnectors = true };
Port M6 = new Port() { Content = "M6", Position = new Point(500, 650), UseSecondConnectors = true };
Port M7 = new Port() { Content = "M7", Position = new Point(500, 700), UseSecondConnectors = true };
Port M8 = new Port() { Content = "M8", Position = new Point(500, 750), UseSecondConnectors = true };
Port M9 = new Port() { Content = "M9", Position = new Point(500, 800), UseSecondConnectors = true };
Port M10 = new Port() { Content = "M10", Position = new Point(500, 850), UseSecondConnectors = true };

RF200.AddItem(M1);
RF200.AddItem(M2);
RF200.AddItem(M3);
RF200.AddItem(M4);
RF200.AddItem(M5);
```

```
RF200.AddItem(M6);
RF200.AddItem(M7);
RF200.AddItem(M8);
RF200.AddItem(M9);
RF200.AddItem(M10);
Link Site1ToM1 = new Link() { Source = Site1, Target = M1 };

Attenuator Atten1 = new Attenuator() { Content = "Attenuator", Position = new Point(375, 500) };
Attenuator Atten2 = new Attenuator() { Content = "Attenuator", Position = new Point(375, 550) };
Attenuator Atten3 = new Attenuator() { Content = "Attenuator", Position = new Point(375, 600) };

this.AddNode(RF100);
this.AddNode(RF200);
this.AddNode(Device);

this.AddNode(Atten1);
this.AddNode(Atten2);
this.AddNode(Atten3);

this.AddLink(P1ToSite1);
this.AddLink(Site1ToM1);
}

public override string GetNodeUniqueId(NodeViewModelBase node)
{
return node.GetHashCode().ToString();
}
}

//public class ConnectorProxy
//{
//    public string Name { get; set; }
//    public System.Windows.Point Position { get; set; }
//}
//
//public static class AttachedProperties
//{
//    public static IEnumerable<ConnectorProxy> GetConnectors(DependencyObject obj)
//    {
//        return (IEnumerable<ConnectorProxy>)obj.GetValue(ConnectorsProperty);
//    }
//
```

```
//    public static void SetConnectors(DependencyObject obj, IEnumerable<ConnectorProxy>
value)
//    {
//        obj.SetValue(ConnectorsProperty, value);
//    }
//
//    public static readonly DependencyProperty ConnectorsProperty =
//        DependencyProperty.RegisterAttached("Connectors",
typeof(IEnumerable<ConnectorProxy>), typeof(AttachedProperties), new PropertyMetadata(null,
OnConnectorsChanged));
//
//    private static void OnConnectorsChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
//    {
//        var shape = d as RadDiagramShape;
//        // You can also check if the collection is INotifyCollectionChanged if you want to change the
connectors runtime.
//        // var connectors = e.NewValue as INotifyCollectionChanged
//        var connectors = e.NewValue as ObservableCollection<ConnectorProxy>;
//        if (shape != null && connectors != null)
//        {
//            foreach (var item in connectors)
//            {
//                //Note: the item.Name should be a valid x:Name
//                shape.Connectors.Add(new RadDiagramConnector() { Name = item.Name, Offset =
item.Position });
//            }
//        }
//    }
//}
}
```

DigitalLevelViewModel.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels
{
public interface IDigitalLevelsView
{
void ForceCommitEdit();
}

public class DigitalLevelViewModel : PVM
{

private IDigitalLevelsView _view;
public IDigitalLevelsView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");

if (value != null)
{
CurrentLevelsData.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}

#region ItemSource Accessors
public List<string> TerminationModes =>  new List<string>() { "High Z", "Vterm", "Active Load" };
public List<string> Senses => new List<string>() { "Local", "Remote" };

public ObservableCollection<Data.PinModel> GetDigitalPins
{
get { return ParentProject.PinMapDataModel.DigitalPins; }
}
#endregion

private object _selectedItem = null;
public object SelectedItem
{
get { return _selectedItem; }
```

```
set { _selectedItem = value; OnPropertyChanged("SelectedItem"); }
}

private DigitalLevelsModel _dataObject = new DigitalLevelsModel("");
public DigitalLevelsModel CurrentLevelsData
{
get { return _dataObject; }
set { _dataObject = value; OnPropertyChanged("CurrentLevelsData"); }
}

#region Constructor
//public event Action CommitEditBeforeSave;
//public event Action ChangeOccured;

public override void PaneClose()
{
this.CurrentLevelsData.CommitEditBeforeSave -= View.ForceCommitEdit;
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
if (this.CurrentLevelsData != null)
{
this.CurrentLevelsData.FilePath = value;
}
}
}
public DigitalLevelViewModel(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;

this.LoadData();
```

```
this.CurrentLevelsData.ChangeOccured += ChangeOccuredSubscriber;
}

public override void SaveData()
{
ForceCommitEdit();
//CommitEditBeforeSave?.Invoke();

this.CurrentLevelsData.Save_NI_XML(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public void ForceCommitEdit()
{
if (View != null)
{
View.ForceCommitEdit();
}
}

public override void LoadData()
{
DigitalLevelsModel dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = DigitalLevelsModel.Load_NI_Xml(this.DataFilePath);
}

if (dataModel != null)
{
this.CurrentLevelsData = dataModel;

this.CurrentLevelsData.FilePath = this.DataFilePath;
this.ParentProject.DigitalPatterns.Add(dataModel);
}

base.LoadData(); //Calls Relink and validate data.
}
```

```
public override void RelinkData()
{
foreach (var Level in this.CurrentLevelsData.DigiLevelsSource)
{
foreach (Data.PinModel pin in this.ParentProject.PinMapDataModel.DigitalPins.Where(pin =>
pin.PinName == Level.PinItem))
{
Level.DigiPin = pin;
}
}
foreach (var Level in this.CurrentLevelsData.PPMULevelsSource)
{
foreach (Data.PinModel pin in this.ParentProject.PinMapDataModel.DigitalPins.Where(pin =>
pin.PinName == Level.PinItem))
{
Level.DigiPin = pin;
}
}
foreach (var Level in this.CurrentLevelsData.DCPowerLevelsSource)
{
foreach (Data.PinModel pin in this.ParentProject.PinMapDataModel.DigitalPins.Where(pin =>
pin.PinName == Level.PinItem))
{
Level.DigiPin = pin;
}
}
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
//public void OnChangeOccured()
//{
//    ChangeOccured?.Invoke();
//}
#endregion


#region Commands

#endregion
}
}
```

DigitalPatternVM.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels
{
public interface IDigitalPatternView
{
//Add columns to the GUI
void AddColumn(string columnName);

void InsertColumn(int index, string columnName);

void RemoveAtColumn(int index);

//Resets the table view
void ResetTable();

void ForceCommitEdit();

void RebindGridView();

void ShowDPatSrcInEditor(string dpatsrcdata);//TEMP
}

public class DigitalPatternVM : PVM
{
public static string UserDialogEntry { get; set; }
```

```
private IDigitalPatternView _view;
public IDigitalPatternView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");

if (value != null)
{
//this.ResetTable();
foreach (string regHeader in this.CurrentPattern.Registers)
{
this.AddColumn(regHeader);
}

OnPropertyChanged("GetTimeSets");
OnPropertyChanged("GetDigitalPins");
CurrentPattern.CommitEditBeforeSave += View.ForceCommitEdit;
}

}
}

#region ItemSource Accessors
public ObservableCollection<Data.PinModel> GetDigitalPins
{
get
{
return ParentProject != null ? ParentProject.PinMapDataModel.DigitalPins : new
ObservableCollection<Data.PinModel>();
}
}

public List<string> Opcodes
{
get { return BackendConstants.RFFE_Opcode_Options; }
}
#endregion
```

```
private object _selectedItem = new Mode(); //{ ModeVectors = new
ObservableCollection<ModeVector>() { new ModeVector()  } }; //null; //default NULL!!! breaks.
public object SelectedItem
{
get
{
if (_selectedItem != null && _selectedItem is Mode)
{
View.ShowDPatSrcInEditor(((Mode)_selectedItem).DPatSrcData);
}
else
{
View.ShowDPatSrcInEditor(" ");
}
return _selectedItem;
}
set
{
if(value != null)
{
_selectedItem = value;
OnPropertyChanged("SelectedItem");
OnPropertyChanged("SelectedVector");
if(((Mode)value).ModeVectors.Count() > 0)
{
SelectedVectorItem = ((Mode)value).ModeVectors[SelectedVector];
SelectedVectorizedData = ((Mode)value).ModeVectors[SelectedVector].VectorizedData;
}

}
}
}

private int _selectedVector = 0;
public int SelectedVector
{
get { return _selectedVector;  }
set
{
_selectedVector = value;
OnPropertyChanged("SelectedVector");
```

```
}
}

private ModeVector _selectedVectorItem = new ModeVector();
public ModeVector SelectedVectorItem
{
get { return _selectedVectorItem; }
set
{
if(value != null)
{
_selectedVectorItem = value;
OnPropertyChanged("SelectedVectorItem");
SelectedVectorizedData = value.VectorizedData;
}
}
}

private ObservableCollection<Operation> _SelectedVectorizedData = new
ObservableCollection<Operation>();
public ObservableCollection<Operation> SelectedVectorizedData
{
get { return _SelectedVectorizedData; }
set
{
if(value != null)
{
_SelectedVectorizedData = value;
OnPropertyChanged("SelectedVectorizedData");
}
}
}

private DigitalPatternModel _dataObject = new DigitalPatternModel("");
public DigitalPatternModel CurrentPattern
{
get { return _dataObject; }
set
{
_dataObject = value;
OnPropertyChanged("CurrentPattern");
}
```

```
}
#region Constructor
//public event Action CommitEditBeforeSave;
//public event Action ChangeOccured;

public override void PaneClose()
{
this.CurrentPattern.CommitEditBeforeSave -= View.ForceCommitEdit;
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
if (this.CurrentPattern != null)
{
this.CurrentPattern.FilePath = value;
}
}
}

public DigitalPatternVM(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;

this.LoadData();

CurrentPattern.ChangeOccured += ChangeOccuredSubscriber;
}

public override void SaveData()
{
ForceCommitEdit();
//CommitEditBeforeSave?.Invoke();
```

```
this.CurrentPattern.SaveToXml(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}
public void ForceCommitEdit()
{
if (View != null)
{
View.ForceCommitEdit();
}
}

public override void LoadData()
{
DigitalPatternModel dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = DigitalPatternModel.LoadFromXml(this.DataFilePath);
}

if (dataModel != null)
{
this.CurrentPattern = dataModel;

#region Backwards Compatible converter (1/~10/2023)
foreach (Mode mode in dataModel.ModeCollection)
{
//convert all mode input values to ModeVectors.
foreach (string str in mode.RegData)
{
mode.ModeVectors.Add(new ModeVector(str));
}
mode.RegData.Clear(); //Clear it after all vectors are added for save/load purposes.
}
#endregion

#region Create operations for empty mode vectors
//Search for modes with no vectorized data to ensure they exist on load for opcode edits.
foreach (Mode mode in this.CurrentPattern.ModeCollection)
{
```

```
mode.FindDetectedOpcodes(); //Check before all the default blanks are added.
foreach (ModeVector vector in mode.ModeVectors)
{
if (vector.VectorizedData.Count() <= 0)
{
this.CurrentPattern.Vectorize_Single(mode, vector);
}
else
{
foreach (Operation op in vector.VectorizedData) //Ensure Original_Input is up to date on load for opcode detection.
{
op.Original_Input = vector.InputValue;
}
}
}
}
#endregion Create operations for empty mode vectors

#region Generate Dpatsrc data for live view
//Compile all modes to ensure DPATSRC data is up to date.
this.CurrentPattern.CompileAllModes();
#endregion

this.CurrentPattern.FilePath = this.DataFilePath;
this.ParentProject.DigitalPatterns.Add(dataModel);
}

base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{
//Actual Relinking of pin data objects
foreach (Data.PinModel pin in this.ParentProject.PinMapDataModel.DigitalPins.Where(pin => pin.PinName == this.CurrentPattern.DataPinItem))
{
this.CurrentPattern.DataPin = pin;
}
foreach (Data.PinModel pin in this.ParentProject.PinMapDataModel.DigitalPins.Where(pin => pin.PinName == this.CurrentPattern.ClockPinItem))
{
```

```
this.CurrentPattern.ClockPin = pin;
}
foreach (DigiTimeObj dto in this.ParentProject.AvailableTimeSets.Where(ts => ts.Name ==
this.CurrentPattern.TimeSetUsed))
{
this.CurrentPattern.TimeSetUsedObj = dto;
}
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
#endregion

#region Methods
public void ForceGetTimeSets()
{
this.ParentProject.TimeSetsChanged();
}
#endregion

#region Commands

#region Opcode Changed
public ICommand OpcodeSelectedCommand { get { return new
DelegateCommand(UpdateOnOpcode); } }
private void UpdateOnOpcode(object obj)
{
if (_selectedItem != null)
{
((Mode)_selectedItem).FindDetectedOpcodes();

//Compile operation vectors.
((Mode)_selectedItem).DPatSrcData =
this.CurrentPattern.CompileModeToDPATSRC(((Mode)_selectedItem));

View.ShowDPatSrcInEditor(((Mode)_selectedItem).DPatSrcData); //Update text editor.

this.CurrentPattern.OnChangeOccured(); //Save needed due to Opcode being stored in an
operation object.
}
}
#endregion
```

```
#region IsCondensed Toggled
public ICommand IsCondensedChangedCommand { get { return new
DelegateCommand(IsCondensedChanged); } }
private void IsCondensedChanged(object obj)
{
if (_selectedItem != null)
{
//Compile operation vectors.
((Mode)_selectedItem).DPatSrcData =
this.CurrentPattern.CompileModeToDPATSRC(((Mode)_selectedItem));

View.ShowDPatSrcInEditor(((Mode)_selectedItem).DPatSrcData); //Update text editor.
}
}
#endregion

#region Compile to dpatsrc and dpat
public ICommand Compile_To_DPAT_Command { get { return new
DelegateCommand(Compile_To_DPAT); } }
private void Compile_To_DPAT(object obj)
{
this.CurrentPattern.GeneratePatternFiles(this.CurrentPattern.FilePath,
ParentProject.PinMapFilePath);
}
#endregion

#region Add Register
public ICommand AddRegisterCommand { get { return new DelegateCommand(AddRegister); } }
private void AddRegister(object obj)
{
RadWindow.Prompt("New register name?", this.OnClosed, "");
if (UserDialogEntry != null) //NO HEX VALUE PROTECTION !!! NEED
{
string regHeader = UserDialogEntry;

CurrentPattern.Registers.Add(regHeader); //Add to backend.

//Adds the Mode wrapper object to each test for value binding.
foreach (Mode m in CurrentPattern.ModeCollection)
{
var vector = new ModeVector("0x00");
m.ModeVectors.Add(vector);
```

```
this.CurrentPattern.Vectorize_Single(m, vector);
}

this.AddColumn(regHeader); //Add to GUI.

this.CurrentPattern.CompileAllModes(); //TEMP //Or build as viewed.
View.ShowDPatSrcInEditor(((Mode)_selectedItem).DPatSrcData); //Update text editor for the
currently selected item.

this.CurrentPattern.OnChangeOccured();
}
}
#endregion

#region Rename Register
public ICommand RenameRegisterCommand { get { return new
DelegateCommand(RenameRegister); } }
private void RenameRegister(object obj)
{
if(obj != null)
{
string oldRegName = (string)obj;
int RegIndex = CurrentPattern.Registers.IndexOf(oldRegName);

RadWindow.Prompt("Change register name", this.OnClosed, oldRegName);
if (UserDialogEntry != null)
{
string regHeader = UserDialogEntry;

CurrentPattern.Registers.RemoveAt(RegIndex); //Remove from back end
CurrentPattern.Registers.Insert(RegIndex, regHeader);

View.RemoveAtColumn(RegIndex); //Rmeove old column from GUI
View.InsertColumn(RegIndex, regHeader); //Insert new column to GUI.

foreach (Mode mode in CurrentPattern.ModeCollection) //Revectorize the data with the new
register renamed. (Will lose existing opcodes)
{
this.CurrentPattern.Vectorize_Single(mode.ModeVectors[RegIndex], RegIndex);
mode.FindDetectedOpcodes();
}
```

```
this.CurrentPattern.CompileAllModes(); //TEMP //Or build as viewed.
View.ShowDPatSrcInEditor(((Mode)_selectedItem).DPatSrcData); //Update text editor for the
currently selected item.

this.CurrentPattern.OnChangeOccured();
}
}

}
#endregion

#region Remove Register
public ICommand RemoveRegisterCommand { get { return new
DelegateCommand(RemoveRegister); } }
private void RemoveRegister(object obj)
{
if(obj != null)
{
string regHeader = obj as string;
int rIndex = CurrentPattern.Registers.IndexOf(regHeader);

foreach(Mode mode in CurrentPattern.ModeCollection) //Removes the Data at the specified
Register index.
{
mode.ModeVectors.RemoveAt(rIndex);
mode.FindDetectedOpcodes();
}

CurrentPattern.Registers.Remove(regHeader); //Removes the back end register.

this.CurrentPattern.CompileAllModes(); //TEMP //Or build as viewed.
View.ShowDPatSrcInEditor(((Mode)_selectedItem).DPatSrcData); //Update text editor for the
currently selected item.

this.CurrentPattern.OnChangeOccured();
}
}
#endregion

#region Add Mode
public ICommand AddModeCommand { get { return new DelegateCommand(AddMode); } }
private void AddMode(object obj)
```

```
{
var regData = new ObservableCollection<ModeVector>();

for (int i = 0; i < CurrentPattern.Registers.Count; i++)
{
var vector = new ModeVector("0x00");
this.CurrentPattern.Vectorize_Single(vector, i);
regData.Add(vector);
}

var modeToAdd = new Mode() { Name = "MODE_NAME", ModeVectors = regData };

this.CurrentPattern.Compile_Single_Mode_(modeToAdd); //TEMP. //Or build as viewed.

this.CurrentPattern.ModeCollection.Add(modeToAdd);

this.CurrentPattern.OnChangeOccured();
}
#endregion

#region Remove Mode
public ICommand RemoveModeCommand { get { return new DelegateCommand(RemoveMode); }
}
private void RemoveMode(object obj)
{
if(obj != null && obj is Mode)
{
CurrentPattern.ModeCollection.Remove(SelectedItem as Mode);

this.CurrentPattern.OnChangeOccured();
}
}
#endregion
#endregion

public void AddColumn(string columnName)
{
View.AddColumn(columnName);
}

public void ResetTable()
{
```

```
View.ResetTable();
}

/// <summary>
/// Calls the Rebind method built-in to the GridView through the code behind.
/// </summary>
private void CallRebindGridView()
{
View.RebindGridView();
}

public void OnClosed(object sender, WindowClosedEventArgs e)
{
UserDialogEntry = null;
if (e.PromptResult != null)
{
UserDialogEntry = e.PromptResult.ToString();
}
}

}
}
```

DigitalTimingVM.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels
{
public interface IDigitalTimingView
{
```

```
void ForceCommitEdit();
}

public class DigitalTimingVM : PVM
{
private IDigitalTimingView _view;
public IDigitalTimingView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");

if (value != null)
{
this.CurrentTimingData.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}

#region ItemSource Accessors
public List<string> EdgeMultipliers => new List<string>() { "1x", "2x" };
public List<string> DriveFormats => new List<string>() { "NR", "RL", "RH", "SBC" };

public ObservableCollection<Data.PinModel> GetDigitalPins
{
get { return ParentProject.PinMapDataModel.DigitalPins; }
}
#endregion

private DigiTimeObj _selectedItem = null;
public DigiTimeObj SelectedItem
{
get { return _selectedItem; }
set { _selectedItem = value; OnPropertyChanged("SelectedItem"); }
}

private DigitalTimingModel _dataObject = new DigitalTimingModel("");
public DigitalTimingModel CurrentTimingData
{
get { return _dataObject; }
```

```
set { _dataObject = value; OnPropertyChanged("CurrentTimingData"); }
}

#region Constructor
//public event Action CommitEditBeforeSave;
//public event Action ChangeOccured;

public override void PaneClose()
{
this.CurrentTimingData.CommitEditBeforeSave -= View.ForceCommitEdit;
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
if(this.CurrentTimingData != null)
{
this.CurrentTimingData.FilePath = value;
}
}
}

public DigitalTimingVM(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;

this.LoadData();

this.CurrentTimingData.ChangeOccured += ChangeOccuredSubscriber;
}

public override void SaveData()
{
ForceCommitEdit();
```

```
//CommitEditBeforeSave?.Invoke();

this.CurrentTimingData.Save_NI_XML(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));
this.ParentProject.TimeSetsChanged();

base.SaveData();
}

public void ForceCommitEdit()
{
if (View != null)
{
View.ForceCommitEdit();
}
}

public override void LoadData()
{
DigitalTimingModel dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = DigitalTimingModel.Load_NI_Xml(this.DataFilePath);
}

if (dataModel != null)
{
this.CurrentTimingData = dataModel;

this.CurrentTimingData.FilePath = this.DataFilePath;
this.ParentProject.DigitalPatterns.Add(dataModel);
this.ParentProject.TimeSetsChanged();
}

base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{
foreach (var digiTime in this.CurrentTimingData.DigiTimeSource)
{
```

```
foreach (Data.PinModel pin in this.ParentProject.PinMapDataModel.DigitalPins.Where(pin =>
pin.PinName == digiTime.PinItem))
{
digiTime.DigiPin = pin;
}
}
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
//public void OnChangeOccured()
//{
//    ChangeOccured?.Invoke();
//}
#endregion

#region Commands

public ICommand InsertItemCommand { get { return new DelegateCommand(InsertItem); } }
private void InsertItem(object obj)
{
try
{
DigiTimeObj objToInsert = new DigiTimeObj() { Name = "DefaultTimeSet" };
if (SelectedItem != null)
{
int indexOfSelected = this.CurrentTimingData.DigiTimeSource.IndexOf(SelectedItem);
this.CurrentTimingData.DigiTimeSource.Insert(indexOfSelected, objToInsert);
}
else
{
this.CurrentTimingData.DigiTimeSource.Add(objToInsert);
}
this.CurrentTimingData.OnChangeOccured();
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
}

public ICommand RemoveItemCommand { get { return new DelegateCommand(RemoveItem); } }
private void RemoveItem(object obj)
```

```
{
try
{
if (SelectedItem != null)
{
this.CurrentTimingData.DigiTimeSource.Remove(SelectedItem);

this.CurrentTimingData.OnChangeOccured();
}
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
}

public ICommand DuplicateItemCommand { get { return new DelegateCommand(DuplicateItem); }
}
private void DuplicateItem(object obj)
{
try
{
if (SelectedItem != null)
{
//Console.WriteLine("No Duplicate for you...");
}
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
}

#endregion Commands
}
}
```

GenericRffePatternVM.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MT.TestStudio.GUI.ViewModels;
```

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels
{
public interface IGenericRffePatternView
{
//Add columns to the GUI
//void AddColumn(string columnName);

//void InsertColumn(int index, string columnName);

//void RemoveAtColumn(int index);

//Resets the table view
void ResetTable();

void ForceCommitEdit();

void RebindGridView();

void RebindDigitalPatternsGridView();

void ShowDPatSrcInEditor(string dpatsrcdata);//TEMP
}

public class GenericRffePatternVM : PVM
{
public static string UserDialogEntry { get; set; }

private IGenericRffePatternView _view;
```

```
public IGenericRffePatternView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");

if (value != null)
{
//this.ResetTable();
///foreach (string regHeader in this.CurrentPattern.Registers)
///{
///    this.AddColumn(regHeader);
///}

OnPropertyChanged("GetTimeSets");
OnPropertyChanged("GetDigitalPins");
View.RebindDigitalPatternsGridView();
CurrentPattern.CommitEditBeforeSave += View.ForceCommitEdit;
}

}
}

#region ItemSource Accessors
public List<KeyValuePair<RffePatternSpec, string>> PatternSpecKvPs
{
get { return BackendConstants.RffePatternSpecKvPs; }
}

public ObservableCollection<Data.PinModel> GetDigitalPins
{
get
{
return ParentProject != null ? ParentProject.PinMapDataModel.DigitalPins : new
ObservableCollection<Data.PinModel>();
}
}

public List<string> Opcodes
{
```

```
get { return BackendConstants.RFFE_Opcode_Options; }
}
#endregion

private object _selectedItem = new GenericMode(); //{ ModeVectors = new
ObservableCollection<ModeVector>() { new ModeVector()  } }; //null; //default NULL!!! breaks.
public object SelectedItem //Needs context fix to select the parent row whenever a child grid row is
selected.
{
get
{
if (_selectedItem != null && _selectedItem is GenericMode)
{
View.ShowDPatSrcInEditor(((GenericMode)_selectedItem).DPatSrcData);
}
else
{
View.ShowDPatSrcInEditor(" ");
}
return _selectedItem;
}
set
{
if(value != null)
{
_selectedItem = value;
OnPropertyChanged("SelectedItem");
OnPropertyChanged("SelectedVector");
if(((GenericMode)value).ModeVectors.Count() > 0)
{
SelectedVectorItem = ((GenericMode)value).ModeVectors[SelectedVector];
SelectedVectorizedData = ((GenericMode)value).ModeVectors[SelectedVector].VectorizedData;
}

}
}
}

private int _selectedVector = 0;
public int SelectedVector
{
get { return _selectedVector;  }
```

```
set
{
_selectedVector = value;
OnPropertyChanged("SelectedVector");
}
}

private GenericModeVector _selectedVectorItem = new GenericModeVector();
public GenericModeVector SelectedVectorItem
{
get { return _selectedVectorItem; }
set
{
if(value != null)
{
_selectedVectorItem = value;
OnPropertyChanged("SelectedVectorItem");
SelectedVectorizedData = value.VectorizedData;
}
}
}

private ObservableCollection<Operation> _SelectedVectorizedData = new
ObservableCollection<Operation>();
public ObservableCollection<Operation> SelectedVectorizedData
{
get { return _SelectedVectorizedData; }
set
{
if(value != null)
{
_SelectedVectorizedData = value;
OnPropertyChanged("SelectedVectorizedData");
}
}
}

private GenericRffePatternModel _dataObject = new GenericRffePatternModel("");
public GenericRffePatternModel CurrentPattern
{
get { return _dataObject; }
set
```

```
{
_dataObject = value;
OnPropertyChanged("CurrentPattern");
}
}


//public int MaxModeVectorsCount
//{
//    get { return this.CurrentPattern.ModeCollection.Count == 0 ? 0 :
CurrentPattern.ModeCollection.Max((x) => x.ModeVectors.Count); }
//}
public List<string> RegisterHeaders = new List<string>();


#region Constructor
//public event Action CommitEditBeforeSave;
//public event Action ChangeOccured;

public override void PaneClose()
{
this.CurrentPattern.CommitEditBeforeSave -= View.ForceCommitEdit;
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
if (this.CurrentPattern != null)
{
this.CurrentPattern.FilePath = value;
}
}
}

public GenericRffePatternVM(Type contentType, string dataFilePath, MerlinProject parentProject,
```

```
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;

this.LoadData();

CurrentPattern.ChangeOccured += ChangeOccuredSubscriber;
}

public override void SaveData()
{
ForceCommitEdit();
//CommitEditBeforeSave?.Invoke();

this.CurrentPattern.SaveToXml(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}
public void ForceCommitEdit()
{
if (View != null)
{
View.ForceCommitEdit();
}
}

public override void LoadData()
{
GenericRffePatternModel dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = GenericRffePatternModel.LoadFromXml(this.DataFilePath);
}

if (dataModel != null)
{
this.CurrentPattern = dataModel;

#region Backwards Compatible converter (1/~10/2023)
foreach (GenericMode mode in dataModel.ModeCollection)
```

```
{
//convert all mode input values to ModeVectors.
foreach (string str in mode.RegData)
{
mode.ModeVectors.Add(new GenericModeVector(str));
}
mode.RegData.Clear(); //Clear it after all vectors are added for save/load purposes.
}
#endregion

#region Create operations for empty mode vectors
//Search for modes with no vectorized data to ensure they exist on load for opcode edits.
foreach (GenericMode mode in this.CurrentPattern.ModeCollection)
{
mode.FindDetectedOpcodes(); //Check before all the default blanks are added.
foreach (GenericModeVector vector in mode.ModeVectors)
{
if (vector.VectorizedData.Count() <= 0)
{
this.CurrentPattern.Vectorize_Single(mode, vector);
}
else
{
foreach (Operation op in vector.VectorizedData) //Ensure Original_Input is up to date on load for opcode detection.
{
op.Original_Input = vector.InputValue;
}
}
}
}
#endregion Create operations for empty mode vectors

#region Generate Dpatsrc data for live view
//Compile all modes to ensure DPATSRC data is up to date.
this.CurrentPattern.CompileAllModes();
#endregion

this.CurrentPattern.FilePath = this.DataFilePath;
this.ParentProject.DigitalPatterns.Add(dataModel);
}
```

```
base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{
//Actual Relinking of pin data objects
foreach (Data.PinModel pin in this.ParentProject.PinMapDataModel.DigitalPins.Where(pin =>
pin.PinName == this.CurrentPattern.DataPinItem))
{
this.CurrentPattern.DataPin = pin;
}
foreach (Data.PinModel pin in this.ParentProject.PinMapDataModel.DigitalPins.Where(pin =>
pin.PinName == this.CurrentPattern.ClockPinItem))
{
this.CurrentPattern.ClockPin = pin;
}
foreach (DigiTimeObj dto in this.ParentProject.AvailableTimeSets.Where(ts => ts.Name ==
this.CurrentPattern.TimeSetUsed))
{
this.CurrentPattern.TimeSetUsedObj = dto;
}
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
#endregion

#region Methods
public void ForceGetTimeSets()
{
this.ParentProject.TimeSetsChanged();
}
#endregion

#region Commands

#region Opcode Changed
public ICommand OpcodeSelectedCommand { get { return new
DelegateCommand(UpdateOnOpcode); } }
private void UpdateOnOpcode(object obj)
{
if (_selectedItem != null)
{
```

```
((GenericMode)_selectedItem).FindDetectedOpcodes();

//Compile operation vectors.
((GenericMode)_selectedItem).DPatSrcData =
this.CurrentPattern.CompileModeToDPATSRC(((GenericMode)_selectedItem));

View.ShowDPatSrcInEditor(((GenericMode)_selectedItem).DPatSrcData); //Update text editor.

this.CurrentPattern.OnChangeOccured(); //Save needed due to Opcode being stored in an
operation object.
}
}
#endregion

#region IsCondensed Toggled
public ICommand IsCondensedChangedCommand { get { return new
DelegateCommand(IsCondensedChanged); } }
private void IsCondensedChanged(object obj)
{
if (_selectedItem != null)
{
//Compile operation vectors.
((GenericMode)_selectedItem).DPatSrcData =
this.CurrentPattern.CompileModeToDPATSRC(((GenericMode)_selectedItem));

View.ShowDPatSrcInEditor(((GenericMode)_selectedItem).DPatSrcData); //Update text editor.

}
}
#endregion

#region Compile to dpatsrc and dpat
public ICommand Compile_To_DPAT_Command { get { return new
DelegateCommand(Compile_To_DPAT); } }
private void Compile_To_DPAT(object obj)
{
this.CurrentPattern.GeneratePatternFiles(this.CurrentPattern.FilePath,
ParentProject.PinMapFilePath);
}
#endregion

#region Add Data To All
```

```
public ICommand AddRegisterCommand { get { return new DelegateCommand(AddRegister); } }
private void AddRegister(object obj)
{
//Adds the Mode wrapper object to each test for value binding.
foreach (GenericMode m in CurrentPattern.ModeCollection)
{
var vector = new GenericModeVector("00");
m.ModeVectors.Add(vector);
this.CurrentPattern.Vectorize_Single(m, vector);
}

this.CurrentPattern.CompileAllModes(); //TEMP //Or build as viewed.
View.ShowDPatSrcInEditor(((GenericMode)_selectedItem).DPatSrcData); //Update text editor for the currently selected item.

this.CurrentPattern.OnChangeOccured();
}
#endregion


#region Add Mode
public ICommand AddModeCommand { get { return new DelegateCommand(AddMode); } }
private void AddMode(object obj)
{
var regData = new ObservableCollection<GenericModeVector>();
var vector = new GenericModeVector("00");
this.CurrentPattern.Vectorize_Single(vector);
regData.Add(vector);

var modeToAdd = new GenericMode() { Name = "MODE_NAME", ModeVectors = regData };

this.CurrentPattern.Compile_Single_Mode_(modeToAdd); //TEMP. //Or build as viewed.

this.CurrentPattern.ModeCollection.Add(modeToAdd);

this.CurrentPattern.OnChangeOccured();
}
#endregion

#region Remove Mode
public ICommand RemoveModeCommand { get { return new DelegateCommand(RemoveMode); }
}
```

```
private void RemoveMode(object obj)
{
if(obj != null && obj is GenericMode)
{
CurrentPattern.ModeCollection.Remove(SelectedItem as GenericMode);

this.CurrentPattern.OnChangeOccured();
}
}
#endregion

#region Insert Mode
/// <summary>
/// Adds a new CalPoint to the editor and logs it (Single-Edit).
/// </summary>
public ICommand InsertModeCommand { get { return new DelegateCommand(InsertMode); } }
private void InsertMode(object obj)
{
//List<KeyValuePair<int, object>> itemsInserted = new List<KeyValuePair<int, object>>();
if (obj is ObservableCollection<object>) //Sent by context menu item.
{
ObservableCollection<object> selecteditems = obj as ObservableCollection<object>;
if (selecteditems.Count() <= 0) //No Items selected or source data is empty, insert 1.
{
int insertIndex = 0;
var regData = new ObservableCollection<GenericModeVector>();
var vector = new GenericModeVector("00");
this.CurrentPattern.Vectorize_Single(vector);
regData.Add(vector);

var modeToInsert = new GenericMode() { Name = "MODE_NAME", ModeVectors = regData };

this.CurrentPattern.ModeCollection.Insert(insertIndex, modeToInsert);
//itemsInserted.Add(new KeyValuePair<int, object>(insertIndex, newPoint));
}
else //Many Items selected, insert the required amount.
{
foreach (var item in selecteditems)
{
int insertIndex = this.CurrentPattern.ModeCollection.IndexOf(item as GenericMode) + 1;
var regData = new ObservableCollection<GenericModeVector>();
var vector = new GenericModeVector("00");
```

```
this.CurrentPattern.Vectorize_Single(vector);
regData.Add(vector);

var modeToInsert = new GenericMode() { Name = "MODE_NAME", ModeVectors = regData };

this.CurrentPattern.ModeCollection.Insert(insertIndex, modeToInsert);
//itemsInserted.Add(new KeyValuePair<int, object>(insertIndex, newPoint));
}
}
//CurrentCalConfig.LogInsertAction(itemsInserted);
}

this.CurrentPattern.OnChangeOccured();
}
#endregion

//public GenericModeVector _childGridSelectedGenericModeVector;
//public GenericModeVector ChildGridSelectedGenericModeVector
//{
//    get { return _childGridSelectedGenericModeVector; }
//    set { _childGridSelectedGenericModeVector = value;
OnPropertyChanged("ChildGridSelectedGenericModeVector"); }
//}

#region Remove Data
public ICommand RemoveDataCommand { get { return new DelegateCommand(RemoveData); } }
private void RemoveData(object obj)
{
if (obj != null && obj is GenericModeVector)
{
//TEMP until better way is found, possibly via datacontext of the item row hosting the child grid
view.
GenericModeVector modeVector = obj as GenericModeVector;
GenericMode parentMode = null;
foreach (GenericMode mode in this.CurrentPattern.ModeCollection)
{
foreach (GenericModeVector mv in mode.ModeVectors)
{
if (mv == modeVector)
{
parentMode = mode;
}
```

```
}
}

if (parentMode.ModeVectors.Contains(modeVector))
{
parentMode.ModeVectors.Remove(modeVector);
}
else
{
//if parent mode is null, this will cause errors.
MessageBox.Show($"{parentMode.Name} does not contain the data being removed, parent mode may be not selected");
Console.WriteLine($"{parentMode.Name} does not contain the data being removed, parent mode may be not selected");
}

this.CurrentPattern.Compile_Single_Mode_(parentMode); //TEMP. //Or build as viewed.

this.CurrentPattern.OnChangeOccured();
}
}
#endregion

#region Add Mode Vector
public ICommand AddDataCommand { get { return new DelegateCommand(AddData); } }
private void AddData(object obj)
{
if (obj != null && obj is GenericModeVector)
{
////Find the parent generic mode
//foreach(GenericMode gm in this.CurrentPattern.ModeCollection )
//{
//
//}

(obj as GenericMode).ModeVectors.Add(new GenericModeVector("00"));

this.CurrentPattern.OnChangeOccured();
}
}
#endregion
```

```
#region Insert Mode Vector
public ICommand InsertDataCommand { get { return new DelegateCommand(InsertData); } }
private void InsertData(object obj)
{
//List<KeyValuePair<int, object>> itemsInserted = new List<KeyValuePair<int, object>>();
if (obj is GenericModeVector) //Sent by context menu item.
{
GenericModeVector selecteditem = obj as GenericModeVector;
GenericMode parentMode = null;
foreach (GenericMode mode in this.CurrentPattern.ModeCollection)
{
foreach (GenericModeVector mv in mode.ModeVectors)
{
if (mv == selecteditem)
{
parentMode = mode;
}
}
}

if (parentMode.ModeVectors.Count() == 0) //No Items selected or source data is empty, insert 1.
{
int insertIndex = 0;
GenericModeVector newData = new GenericModeVector("00");
this.CurrentPattern.Vectorize_Single(newData);

parentMode.ModeVectors.Insert(insertIndex, newData);
}
else
{

int insertIndex = parentMode.ModeVectors.IndexOf(selecteditem as GenericModeVector) + 1;
GenericModeVector newData = new GenericModeVector("00");
this.CurrentPattern.Vectorize_Single(newData);

parentMode.ModeVectors.Insert(insertIndex, newData);
}
}

this.CurrentPattern.OnChangeOccured();
}
#endregion
```

```
#endregion

///public void AddColumn(string columnName)
///{
///    View.AddColumn(columnName);
///}

public void ResetTable()
{
View.ResetTable();
}

/// <summary>
/// Calls the Rebind method built-in to the GridView through the code behind.
/// </summary>
private void CallRebindGridView()
{
View.RebindGridView();
}

public void OnClosed(object sender, WindowClosedEventArgs e)
{
UserDialogEntry = null;
if (e.PromptResult != null)
{
UserDialogEntry = e.PromptResult.ToString();
}
}

}
}
```

ConditionsViewModel.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.TestModels;
using MerlinTestStudio_Demo_Telerik.Data.Services;
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
```

```
using System.Data;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.Windows;
using System.Windows.Input;
using System.Xml.Serialization;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Spreadsheet.Worksheets.Layers;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
//public interface IDynamicColumns
//{
//    void ConsumeColumn();
//}
public interface IConditionsView // : IDynamicColumns
{
void ForceCommitEdit();
//Adds a column to the conditions table
void AddColumn(string columnName, string unit);

//Resets the table view
void ResetTable();

void RebindGridView();
}

public class ConditionsViewModel : PVM  //IPVMLink
{
[XmlIgnore]
public static string UserDialogEntry { get; set; }

private IConditionsView _view;
[XmlIgnore]
public IConditionsView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");
```

```
if (value != null)
{
this.CommitEditBeforeSave += View.ForceCommitEdit; //Won't work as data object is not set, just an example.

//Auto Generates the columns manually to avoid duplicates.
foreach (Data.ParameterMapTag pmt in ParentProject.ParameterMapTags)
AddColumn(pmt.ColumnName, pmt.Unit);
}

}
}
//private PaneViewModel _pvm;
//[XmlIgnore]
//public PaneViewModel PVM
//{
//    get { return _pvm; }
//    set
//    {
//        _pvm = value;
//        OnPropertyChanged("PVM");
//        //Auto Generates the columns manually to avoid duplicates.
//        foreach (Data.ParameterMapTag pmt in value.ParentProject.ParameterMapTags)
//            AddColumn(pmt.ColumnName, pmt.Unit);
//    }
//}

[XmlIgnore]
public ObservableCollection<Data.ParameterMapTag> ParameterTags
{
get { return ParentProject.ParameterMapTags; }
}
[XmlIgnore]
public ObservableCollection<Test> TestCollection
{
get { return ParentProject.Tests; }
}

private Test _selectedTest;
[XmlIgnore]
public Test SelectedTest
```

```
{
get { return _selectedTest; }
set
{
if(value != null)
{
_selectedTest = value;
OnPropertyChanged("SelectedTest");
}
}
}


#region Constructor
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;
public override void PaneClose()
{
this.View = null;
}

public event Action<string> OnChangesMade = (UserControl) => { };

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
//Header = Path.GetFileName(value);
}
}

/// <summary>
/// Constructor for this ViewModel
/// </summary>
public ConditionsViewModel(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;
```

```
this.LoadData();

OnChangesMade += (UserControl) =>
{
ParentProject.RenumerateTestCollection();

//NEED to switch all usages to OnChangeOccurred!
if (UserControl == "DUT Test") //TEMP CODE
{
((PVM)ParentProject.ProjectTree[3].FolderItems[0]).IsSaveRequired = true;
}
else
{
IsSaveRequired = true;
}
};

UpOrDownCommand = new DelegateCommand(UpOrDown);
MoveSelectedToCommand = new DelegateCommand(MoveSelectedTo);

this.ChangeOccured += ChangeOccuredSubscriber;
}

public override void SaveData()
{
CommitEditBeforeSave?.Invoke();

ParentProject.SaveParameterMapTagData(ParentProject.ParameterMapTagsFilePath);

ParentProject.RenumerateTestCollection(); //Test Break Tolerant.

ParentProject.SaveTestItemsData(ParentProject.TestParametersFilePath);

Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public override void LoadData()
{
MerlinProject.LoadParameterMapTagData(ParentProject.ParameterMapTagsFilePath,
```

```
ParentProject);
MerlinProject.LoadTestItemsData(ParentProject.TestParametersFilePath, ParentProject);

base.LoadData();
}

public override void RelinkData()
{
ParentProject.RelinkInternalData();
base.RelinkData();
}

public override void ValidateData()
{
base.ValidateData();
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
#endregion


#region Cell Edited Command

public ICommand CellEditedCommand { get { return new RelayCommand(CellEdited); } }
private void CellEdited()
{
OnChangesMade("Test Parameters");
}

#endregion

#region RenumerateTests Command

public ICommand RenumerateTestsCommand { get { return new
RelayCommand(RenumerateTests); } }
private void RenumerateTests()
{
//Renumerates the test number property for each item in test collection.
```

```
//_testService.RenumerateTestCollection();

OnChangesMade("Test Parameters");
}

#endregion

#region AddTest Command
public ICommand AddTestCommand { get { return new RelayCommand(AddTest); } }
private void AddTest()
{
ParentProject.AddNewTest();

OnChangesMade("Test Parameters");
OnChangesMade("DUT Test");
}

#endregion

#region InsertTest Command
public ICommand InsertTestCommand { get { return new DelegateCommand(InsertTest); } }
private void InsertTest(object obj)
{
if (obj != null && obj is Test)
{
var test = obj as Test;
int index;
MessageBoxResult messageBoxResult = System.Windows.MessageBox.Show("Insert(Yes)  or
Add(No)?", $"Insert at test number {test.TestNumber} - {test.TestName}",
System.Windows.MessageBoxButton.YesNo);

//Process user input to determine the action to be taken.
switch (messageBoxResult)
{
case MessageBoxResult.Yes: index = ParentProject.Tests.IndexOf(test); break; //Insert new test.
case MessageBoxResult.No: index = -1; break; //Append new test.
default: index = -2; break; //Cancelled
}

if(index != -2) //If index is -1 a new test will be appended, any value above -1 will add new test to
the currently selected index.
{
```

```
ParentProject.InsertNewTest(index);

OnChangesMade("Test Parameters");
OnChangesMade("DUT Test");
}
}
}

#endregion

#region RemoveTest Command
public ICommand RemoveTestCommand { get { return new DelegateCommand(RemoveTest); } }
private void RemoveTest(object obj)
{
if (obj != null && obj is Test)
{
var test = obj as Test;
MessageBoxResult messageBoxResult = System.Windows.MessageBox.Show("Are you sure?",
$"Delete Test Number {test.TestNumber} - {test.TestName}",
System.Windows.MessageBoxButton.YesNo);
if (messageBoxResult == MessageBoxResult.Yes)
{
ParentProject.RemoveTest(obj as Test);

OnChangesMade("Test Parameters");
OnChangesMade("DUT Test");
}
}
}
#endregion

#region DuplicateTest Command
public ICommand DuplicateTestCommand { get { return new DelegateCommand(DuplicateTest); }
}
private void DuplicateTest(object obj)
{
if(obj != null && obj is ITest)
{
ParentProject.DuplicateTest(obj as Test);

OnChangesMade("Test Parameters");
OnChangesMade("DUT Test");
```

```
}
}
#endregion

#region DesignateTestNumerationBreak Command

public ICommand DesignateTestNumerationBreakCommand { get { return new
DelegateCommand(DesignateTestNumerationBreak); } }
private void DesignateTestNumerationBreak(object obj)
{
if(obj != null && obj is ITest)
{
var test = obj as ITest;
test.IsTestNumBreak = test.IsTestNumBreak ? false : true; //Reverese the bool value.
OnChangesMade("Test Parameters");
}
}

#endregion



#region Up/Down Command

public ICommand UpOrDownCommand { get; set; }

/// <summary>
/// Moves the selected Group Up or Down by 1.
/// </summary>
/// <param name="obj"></param>
private void UpOrDown(object obj)
{
if (SelectedTest != null && obj != null)
{
int currentIndex = ParentProject.Tests.IndexOf(SelectedTest);
int newIndex = obj.ToString() == "Up" ? currentIndex - 1 :
obj.ToString() == "Down" ? currentIndex + 1 : currentIndex; //Default uses the current index.

if (newIndex < 0 || newIndex > ParentProject.Tests.Count - 1) //If the new index for item insertion
is out of bounds, return.
return;

ParentProject.Tests.Move(currentIndex, newIndex);
```

```
OnChangesMade("Test Parameters");
}
}

#endregion

#region MoveSelectedToCommand

public ICommand MoveSelectedToCommand { get; set; }

private void MoveSelectedTo(object obj)
{
if (SelectedTest != null && obj != null)
{
int oldIndex = ParentProject.Tests.IndexOf(SelectedTest);
int newIndex = int.Parse(obj.ToString()) - 1;

ParentProject.Tests.Move(oldIndex, newIndex);
OnChangesMade("Test Parameters");
}
}

#endregion

//Needs Control and Content
#region OpenProductInfo Command

public ICommand OpenProductInfoCommand { get { return new
RelayCommand(OpenProductInfo); } }
private void OpenProductInfo()
{
//Opens Product information window for viewing only
RadWindow ProductInfo = new RadWindow()
{
Header = "Product Information",
WindowStartupLocation = System.Windows.WindowStartupLocation.CenterOwner,
Width = 400,
Height = 300,
Content = null
};
ProductInfo.ShowDialog();
}
```

```
#endregion


#region AddTestParameter Command

public ICommand AddTestParameterCommand { get { return new
RelayCommand(AddTestParameter); } }
private void AddTestParameter()
{
RadWindow.Prompt("New parameter name?", this.OnClosed, "");
if (UserDialogEntry != null)
{
var ParamMapTag = new ParameterMapTag()
{
IsAutoGen = false,
ColumnIndex = ParentProject.ParameterMapTags.Count,
AttributeType = TestAttributeType.Parameter,
Unit = "",
ColumnName = UserDialogEntry
};

//Adds the TestParameter wrapper object to each test for value binding.
foreach (ITest t in ParentProject.Tests)
t.Parameters.Add(new TestParameter() { Value = string.Empty });

ParentProject.AddColumn(UserDialogEntry, typeof(string), ParamMapTag);
OnChangesMade("Test Parameters");
}
}

#endregion

#region DeleteTestParameter Command

public ICommand DeleteTestParameterCommand { get { return new
DelegateCommand(DeleteTestParameter); } }
private void DeleteTestParameter(object obj)
{
if(obj != null && obj is ParameterMapTag)
{
int paramIndex = ParentProject.ParameterMapTags.IndexOf(obj as ParameterMapTag);
```

```
ParentProject.ParameterMapTags.RemoveAt(paramIndex);

foreach (Test t in ParentProject.Tests)
t.Parameters.RemoveAt(paramIndex);
}

}

#endregion

public void AddColumn(string columnName, string unit)
{
if (View != null)
{
View.AddColumn(columnName, unit);
}
}

public void ResetTable()
{
if (View != null)
{
View.ResetTable();
}
}

/// <summary>
/// Calls the Rebind method built-in to the GridView through the code behind.
/// </summary>
private void CallRebindGridView()
{
if(View != null)
{
View.RebindGridView();
}
}

public void OnClosed(object sender, WindowClosedEventArgs e)
{
UserDialogEntry = null;
if (e.PromptResult != null)
```

```
{
UserDialogEntry = e.PromptResult.ToString();
}
}

}
}
```

GoldLimitsVM.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels
{
public interface IGoldLimitsView
{
void ForceCommitEdit();
}

public class GoldLimitsVM : PVM
{
private IGoldLimitsView _view;
private GoldLimitsData _testLimitsObj = new GoldLimitsData() { Data = new List<GoldLimit>() };
private GoldLimit _selectedLimit;


public IGoldLimitsView View
{
get { return _view; }
set
{
_view = value;
```

```
OnPropertyChanged("View");
if (value != null)
{
this.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}

public GoldLimitsData GoldLimitsObj
{
get { return _testLimitsObj; }
set { _testLimitsObj = value; OnPropertyChanged("GoldLimitsObj"); }
}

//public ObservableCollection<Test> TestsCollection
//{
//    get { return ParentProject.Tests; }
//}

public GoldLimit SelectedLimit
{
get { return _selectedLimit; }
set
{
if (value != null)
{
_selectedLimit = value;
OnPropertyChanged("SelectedLimit");
}
}
}

#region Constructor
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;

public override void PaneClose()
{
this.View = null;
}

private string _dataFilePath;
```

```
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
}
}
public GoldLimitsVM(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;

this.LoadData();

this.ChangeOccured += ChangeOccuredSubscriber;
}
public override void SaveData()
{
CommitEditBeforeSave?.Invoke();

this.GoldLimitsObj.SaveToXml(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public override void LoadData()
{
GoldLimitsData dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = GoldLimitsData.LoadFromXml(this.DataFilePath);
}

if (dataModel != null)
{
this.GoldLimitsObj = dataModel;
this.ParentProject.TestLimits.Add(dataModel);
```

```
}

base.LoadData();  //Calls Relink and validate data.
}

public override void RelinkData()
{
if (this.ParentProject.BaseProjectTestInfo.Count() == 0)
{
Console.WriteLine("Creating Base Test Data from gold limits file...");
for (int i = 0; i < GoldLimitsObj.Data.Count(); i++)
{
var offsetLimit = GoldLimitsObj.Data[i];
var testDataInfo = new TestDataInfo()
{
TestName = offsetLimit.TestName,
TestNumber = offsetLimit.TestNumber,
TestUnits = offsetLimit.Units
};
this.ParentProject.BaseProjectTestInfo.Add(testDataInfo);
}
}
else if (GoldLimitsObj.Data.Count() == 0)
{
Console.WriteLine("Creating Gold Limit Data from base test info...");
for (int i = 0; i < this.ParentProject.BaseProjectTestInfo.Count(); i++)
{
var testBaseInfo = this.ParentProject.BaseProjectTestInfo[i];
var goldLimit = new GoldLimit()
{
TestName = testBaseInfo.TestName,
TestNumber = testBaseInfo.TestNumber,
Units = testBaseInfo.TestUnits,

GoldRetestControl = GoldControl.Check_To_Retest //Default?
};
GoldLimitsObj.Data.Add(goldLimit);
}
}
//else if (parentProject.BaseProjectTestInfo.Count() > TestFixturesObj.Data.Count()) //Sync limit
data with base test data on load...
//{
```

```
//
//}
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
#endregion

#region Methods

#endregion Methods

#region Commands
public ICommand AddTestGoldLimitCommand { get { return new
DelegateCommand(AddTestGoldLimit); } }
private void AddTestGoldLimit(object obj)
{
try
{
//Do Something

OnChangeOccured();
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
}

#endregion



}
}
```

OffsetLimitsVM.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using System;
using System.Collections.Generic;
```

```
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels
{
public interface IOffsetLimitsView
{
void ForceCommitEdit();

void RebindGridView();

void RebindOffsetLimitsGridView();
}

public class OffsetLimitsVM : PVM
{
private IOffsetLimitsView _view;
private OffsetLimitsData _offsetLimitsObj = new OffsetLimitsData() {Data = new List<OffsetLimit>()
};
private OffsetLimit _selectedLimit;



public IOffsetLimitsView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");
if (value != null)
{
View.RebindOffsetLimitsGridView();
this.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}
```

```
public OffsetLimitsData OffsetLimitsObj
{
get { return _offsetLimitsObj; }
set { _offsetLimitsObj = value; OnPropertyChanged("TestLimitsObj"); }
}

//public ObservableCollection<Test> TestsCollection
//{
//    get { return ParentProject.Tests; }
//}

public OffsetLimit SelectedLimit
{
get { return _selectedLimit; }
set
{
if (value != null)
{
_selectedLimit = value;
OnPropertyChanged("SelectedLimit");
}
}
}

public int MaxOffsetSitesCount
{
get { return OffsetLimitsObj.Data.Count == 0 ? 0 : OffsetLimitsObj.Data.Max((x) =>
x.OffsetSites.Count); }
}
public List<string> OffsetSiteHeaders = new List<string>();

#region Constructor
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;

public override void PaneClose()
{
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
```

```
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
}
}
public OffsetLimitsVM(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;

this.LoadData();

this.ChangeOccured += ChangeOccuredSubscriber;
}
public override void SaveData()
{
CommitEditBeforeSave?.Invoke();

this.OffsetLimitsObj.SaveToXml(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public override void LoadData()
{
OffsetLimitsData dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = OffsetLimitsData.LoadFromXml(this.DataFilePath);
}

if (dataModel != null)
{
this.OffsetLimitsObj = dataModel;
this.ParentProject.TestLimits.Add(dataModel);
}
```

```
base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{
if (this.ParentProject.BaseProjectTestInfo.Count() == 0)
{
Console.WriteLine("Creating Base Test Data from offset limits file...");
for (int i = 0; i < OffsetLimitsObj.Data.Count(); i++)
{
var offsetLimit = OffsetLimitsObj.Data[i];
var testDataInfo = new TestDataInfo()
{
TestName = offsetLimit.TestName,
TestNumber = offsetLimit.TestNumber,
TestUnits = offsetLimit.Units
};
this.ParentProject.BaseProjectTestInfo.Add(testDataInfo);
}
}
else if (OffsetLimitsObj.Data.Count() == 0)
{
Console.WriteLine("Creating Offset Limit Data from base test info...");
for (int i = 0; i < this.ParentProject.BaseProjectTestInfo.Count(); i++)
{
var testBaseInfo = this.ParentProject.BaseProjectTestInfo[i];
var offsetLimit = new OffsetLimit()
{
TestName = testBaseInfo.TestName,
TestNumber = testBaseInfo.TestNumber,
Units = testBaseInfo.TestUnits,

OffsetSites = new List<double>() { 0 }
};
OffsetLimitsObj.Data.Add(offsetLimit);
}
}
//else if (parentProject.BaseProjectTestInfo.Count() > TestFixturesObj.Data.Count()) //Sync limit
data with base test data on load...
//{
//
```

```
//}
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
#endregion

#region Methods

#endregion Methods

#region Commands
public ICommand AddOffsetSiteCommand { get { return new DelegateCommand(AddOffsetSite); }
}
private void AddOffsetSite(object obj)
{
try
{
foreach (var limit in OffsetLimitsObj.Data)
{
limit.OffsetSites.Add(0);
}

View.RebindOffsetLimitsGridView();

OnChangeOccured();
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
}

public ICommand RemoveOffsetSiteCommand { get { return new
DelegateCommand(RemoveOffsetSite); } }
private void RemoveOffsetSite(object obj)
{
try
{
int indexOfBinToRemove = (int)obj;
```

```
foreach (var limit in OffsetLimitsObj.Data)
{
limit.OffsetSites.RemoveAt(indexOfBinToRemove);
}

View.RebindOffsetLimitsGridView();

OnChangeOccured();
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
}

#endregion



}
}
```

TestFixturesVM.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels
{
public interface ITestFixturesView
{

void ForceCommitEdit();

void RebindGridView();
```

```
void RebindOffsetLimitsGridView();
}

public class TestFixturesVM : PVM
{
private ITestFixturesView _view;
private TestFixtureData _limitsObj = new TestFixtureData() { Data = new List<TestFixtureLimit>()
};
private TestFixtureLimit _selectedLimit;


public ITestFixturesView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");
if (value != null)
{
View.RebindOffsetLimitsGridView();
this.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}

public TestFixtureData TestFixturesObj
{
get { return _limitsObj; }
set { _limitsObj = value; OnPropertyChanged("TestFixturesObj"); }
}

//public ObservableCollection<Test> TestsCollection
//{
//    get { return ParentProject.Tests; }
//}

public TestFixtureLimit SelectedLimit
{
get { return _selectedLimit; }
set
{
```

```
if (value != null)
{
_selectedLimit = value;
OnPropertyChanged("SelectedLimit");
}
}
}

public int MaxTestFixtureSitesCount
{
get { return TestFixturesObj.Data.Count == 0 ? 0 : TestFixturesObj.Data.Max((x) =>
x.TestFixtureSites.Count); }
}
public List<string> TestFixtureSiteHeaders = new List<string>();

#region Constructor
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;

public override void PaneClose()
{
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
}
}
public TestFixturesVM(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;

this.LoadData();
```

```
this.ChangeOccured += ChangeOccuredSubscriber;
}
public override void SaveData()
{
CommitEditBeforeSave?.Invoke();

this.TestFixturesObj.SaveToXml(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public override void LoadData()
{
TestFixtureData dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = TestFixtureData.LoadFromXml(this.DataFilePath);
}

if (dataModel != null)
{
this.TestFixturesObj = dataModel;
this.ParentProject.TestLimits.Add(dataModel);
}

base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{
if (this.ParentProject.BaseProjectTestInfo.Count() == 0) //Create test data if needed.
{
Console.WriteLine("Creating Base Test Data from test fixtures file...");
for (int i = 0; i < TestFixturesObj.Data.Count(); i++)
{
var testFixture = TestFixturesObj.Data[i];
var testDataInfo = new TestDataInfo()
{
TestName = testFixture.TestName,
TestNumber = testFixture.TestNumber,
```

```
TestUnits = testFixture.Units
};
this.ParentProject.BaseProjectTestInfo.Add(testDataInfo);
}
}
else if (TestFixturesObj.Data.Count() == 0) //Create limits based on test data if needed.
{
Console.WriteLine("Creating Test Fixture Data from base test info...");
for (int i = 0; i < this.ParentProject.BaseProjectTestInfo.Count(); i++)
{
var testBaseInfo = this.ParentProject.BaseProjectTestInfo[i];
var testFixture = new TestFixtureLimit()
{
TestName = testBaseInfo.TestName,
TestNumber = testBaseInfo.TestNumber,
Units = testBaseInfo.TestUnits,

ApplyTestFixture = OffsetControl.Do_Nothing,
TestFixtureSites = new List<string>() { string.Empty }
};
TestFixturesObj.Data.Add(testFixture);
}
}
//else if (parentProject.BaseProjectTestInfo.Count() > TestFixturesObj.Data.Count()) //Sync limit
data with base test data on load...
//{
//
//}
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
#endregion

#region Methods

#endregion Methods

#region Commands
```

```
public ICommand AddTestFixtureSiteCommand { get { return new
DelegateCommand(AddTestFixtureSite); } }
private void AddTestFixtureSite(object obj)
{
try
{
foreach (var limit in TestFixturesObj.Data)
{
limit.TestFixtureSites.Add(string.Empty);
}

View.RebindOffsetLimitsGridView();

OnChangeOccured();
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
}

public ICommand RemoveTestFixtureSiteCommand { get { return new
DelegateCommand(RemoveTestFixtureSite); } }
private void RemoveTestFixtureSite(object obj)
{
try
{
int indexToRemove = (int)obj;

foreach (var limit in TestFixturesObj.Data)
{
limit.TestFixtureSites.RemoveAt(indexToRemove);
}

View.RebindOffsetLimitsGridView();

OnChangeOccured();
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
}

#endregion
```

```
}
}
```

TestLimitsVM.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Models.TestModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Data;
using System.Windows.Input;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels
{
public interface ITestLimitsView : IDisposable
{
void ForceCommitEdit();

void RebindGridView();

void RebindTestLimitsGridView();
}

public class TestLimitsVM : PVM
{
private ITestLimitsView _view;
private UpperLowerLimitsData _testLimitsObj = new UpperLowerLimitsData() { Data = new
List<UpperLowerLimit>() };
private UpperLowerLimit _selectedLimit;



public ITestLimitsView View
{
```

```
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");
if (value != null)
{
View.RebindTestLimitsGridView();
this.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}

public UpperLowerLimitsData TestLimitsObj
{
get { return _testLimitsObj; }
set { _testLimitsObj = value; OnPropertyChanged("TestLimitsObj"); }
}

//public ObservableCollection<Test> TestsCollection
//{
//    get { return ParentProject.Tests; }
//}

public UpperLowerLimit SelectedLimit
{
get { return _selectedLimit; }
set
{
if (value != null)
{
_selectedLimit = value;
OnPropertyChanged("SelectedLimit");
}
}
}

public int MaxMultiPassBinsCount
{
get { return TestLimitsObj.Data.Count == 0 ? 0 : TestLimitsObj.Data.Max((x) =>
x.MultiPassBins.Count); }
}
```

```
public int MaxMultiFailBinsCount
{
get { return TestLimitsObj.Data.Count == 0 ? 0 : TestLimitsObj.Data.Max((x) =>
x.MultiFailBins.Count); }
}
public List<string> MultiPassHeaders = new List<string>();
public List<string> MultiFailHeaders = new List<string>();



#region Constructor
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;

public override void PaneClose()
{
this.CommitEditBeforeSave -= View.ForceCommitEdit;
this.View.Dispose();
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
}
}
public TestLimitsVM(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;

this.LoadData();

this.ChangeOccured += ChangeOccuredSubscriber;
}

public override void SaveData()
```

```
{
CommitEditBeforeSave?.Invoke();

this.TestLimitsObj.SaveToXml(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public override void LoadData()
{
UpperLowerLimitsData dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = UpperLowerLimitsData.LoadFromXml(this.DataFilePath);
}

if (dataModel != null)
{
this.TestLimitsObj = dataModel;
this.ParentProject.TestLimits.Add(dataModel);
}

base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{
if (this.ParentProject.BaseProjectTestInfo.Count() == 0)
{
Console.WriteLine("Creating Base Test Data from limits file...");
for (int i = 0; i < TestLimitsObj.Data.Count(); i++)
{
var upperLowerLimit = TestLimitsObj.Data[i];
var testDataInfo = new TestDataInfo()
{
TestName = upperLowerLimit.TestName,
TestNumber = upperLowerLimit.TestNumber,
TestUnits = upperLowerLimit.Units,
};
this.ParentProject.BaseProjectTestInfo.Add(testDataInfo);
```

```
}
}
else if (TestLimitsObj.Data.Count() == 0)
{
Console.WriteLine("Creating Limit Data from base test info...");
for (int i = 0; i < this.ParentProject.BaseProjectTestInfo.Count(); i++)
{
var testBaseInfo = this.ParentProject.BaseProjectTestInfo[i];
var limit = new UpperLowerLimit()
{
TestName = testBaseInfo.TestName,
TestNumber = testBaseInfo.TestNumber,
Units = testBaseInfo.TestUnits,

MultiPassBins = new ObservableCollection<MultiPassBin>(),
MultiFailBins = new ObservableCollection<MultiFailBin>()
};
TestLimitsObj.Data.Add(limit);
}
}
//else if (this.ParentProject.BaseProjectTestInfo.Count() > TestFixturesObj.Data.Count()) //Sync
limit data with base test data on load...
//{
//
//}
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
#endregion

#region Methods

#endregion Methods

#region Commands
public ICommand AddMultiPassBinCommand { get { return new
DelegateCommand(AddMultiPassBin); } }
private void AddMultiPassBin(object obj)
```

```
{
try
{
foreach (var upperLowerLimit in TestLimitsObj.Data)
{
upperLowerLimit.MultiPassBins.Add(new MultiPassBin());
}

View.RebindTestLimitsGridView();

OnChangeOccured();
}
catch(Exception ex )
{ MessageBox.Show(ex.Message);  }
}

public ICommand RemoveMultiPassBinCommand { get { return new
DelegateCommand(RemoveMultiPassBin);  } }
private void RemoveMultiPassBin(object obj)
{
try
{
int indexOfBinToRemove = (int)obj;

foreach (var upperLowerLimit in TestLimitsObj.Data)
{
upperLowerLimit.MultiPassBins.RemoveAt(indexOfBinToRemove);
}

View.RebindTestLimitsGridView();

OnChangeOccured();
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
}

public ICommand AddMultiFailBinCommand { get { return new
DelegateCommand(AddMultiFailBin); } }
private void AddMultiFailBin(object obj)
{
try
```

```
{
foreach (var upperLowerLimit in TestLimitsObj.Data)
{
upperLowerLimit.MultiFailBins.Add(new MultiFailBin());
}

View.RebindTestLimitsGridView();

OnChangeOccured();
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
}

public ICommand RemoveMultiFailBinCommand { get { return new
DelegateCommand(RemoveMultiFailBin); } }
private void RemoveMultiFailBin(object obj)
{
try
{
int indexOfBinToRemove = (int)obj;

foreach (var upperLowerLimit in TestLimitsObj.Data)
{
upperLowerLimit.MultiFailBins.RemoveAt(indexOfBinToRemove);
}

View.RebindTestLimitsGridView();

OnChangeOccured();
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
}
#endregion



}
}
```

TracelossLimitsVM.cs

```
using MerlinTest.Common.Types;
```

```
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels
{

public interface ITracelossLimitsView
{

void ForceCommitEdit();

void RebindGridView();

void RebindOffsetLimitsGridView();
}

public class TracelossLimitsVM : PVM
{
private ITracelossLimitsView _view;
private TraceLossData _tracelossLimitsObj = new TraceLossData() { Data = new
List<TraceLossLimit>() };
private TraceLossLimit _selectedLimit;



public ITracelossLimitsView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");
if (value != null)
{
```

```
View.RebindOffsetLimitsGridView();
this.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}

public TraceLossData TracelossLimitsObj
{
get { return _tracelossLimitsObj; }
set { _tracelossLimitsObj = value; OnPropertyChanged("TracelossLimitsObj"); }
}

//public ObservableCollection<Test> TestsCollection
//{
//    get { return ParentProject.Tests; }
//}

public TraceLossLimit SelectedLimit
{
get { return _selectedLimit; }
set
{
if (value != null)
{
_selectedLimit = value;
OnPropertyChanged("SelectedLimit");
}
}
}

public int MaxTracelossSitesCount
{
get { return TracelossLimitsObj.Data.Count == 0 ? 0 : TracelossLimitsObj.Data.Max((x) =>
x.TracelossSites.Count); }
}
public List<string> TracelossSiteHeaders = new List<string>();

#region Constructor
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;

public override void PaneClose()
```

```
{
this.View = null;
}

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
Header = Path.GetFileName(value);
}
}
public TracelossLimitsVM(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
this.DataFilePath = dataFilePath;

this.LoadData();

this.ChangeOccured += ChangeOccuredSubscriber;
}
public override void SaveData()
{
CommitEditBeforeSave?.Invoke();

this.TracelossLimitsObj.SaveToXml(this.DataFilePath);
Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public override void LoadData()
{
TraceLossData dataModel = null;

if (File.Exists(this.DataFilePath))
{
dataModel = TraceLossData.LoadFromXml(this.DataFilePath);
}
```

```
if (dataModel != null)
{
this.TracelossLimitsObj = dataModel;
this.ParentProject.TestLimits.Add(dataModel);
}

base.LoadData(); //Calls Relink and validate data.
}

public override void RelinkData()
{
if (this.ParentProject.BaseProjectTestInfo.Count() == 0)
{
Console.WriteLine("Creating Base Test Data from traceloss limits file...");
for (int i = 0; i < TracelossLimitsObj.Data.Count(); i++)
{
var tracelossLimit = TracelossLimitsObj.Data[i];
var testDataInfo = new TestDataInfo()
{
TestName = tracelossLimit.TestName,
TestNumber = tracelossLimit.TestNumber,
TestUnits = tracelossLimit.Units
};
this.ParentProject.BaseProjectTestInfo.Add(testDataInfo);
}
}
else if (TracelossLimitsObj.Data.Count() == 0)
{
Console.WriteLine("Creating Traceloss Limit Data from base test info...");
for (int i = 0; i < this.ParentProject.BaseProjectTestInfo.Count(); i++)
{
var testBaseInfo = this.ParentProject.BaseProjectTestInfo[i];
var tracelossLimit = new TraceLossLimit()
{
TestName = testBaseInfo.TestName,
TestNumber = testBaseInfo.TestNumber,
Units = testBaseInfo.TestUnits,

ApplyTraceLoss = OffsetControl.Do_Nothing,
TracelossSites = new List<double>() { 0 }
};
```

```
TracelossLimitsObj.Data.Add(tracelossLimit);
}
}
//else if (parentProject.BaseProjectTestInfo.Count() > TestFixturesObj.Data.Count()) //Sync limit
data with base test data on load...
//{
//
//}
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
#endregion

#region Methods

#endregion Methods

#region Commands
public ICommand AddTracelossSiteCommand { get { return new
DelegateCommand(AddTracelossSite); } }
private void AddTracelossSite(object obj)
{
try
{
foreach (var limit in TracelossLimitsObj.Data)
{
limit.TracelossSites.Add(0);
}

View.RebindOffsetLimitsGridView();

OnChangeOccured();
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
}

public ICommand RemoveTracelossSiteCommand { get { return new
```

```
DelegateCommand(RemoveTracelossSite); } }
private void RemoveTracelossSite(object obj)
{
try
{
int indexToRemove = (int)obj;

foreach (var limit in TracelossLimitsObj.Data)
{
limit.TracelossSites.RemoveAt(indexToRemove);
}

View.RebindOffsetLimitsGridView();

OnChangeOccured();
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
}

#endregion



}
}
```

DUT_End_TestViewModel.cs

```
using MT.TestStudio.GUI.ViewModels;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public class DUT_End_TestViewModel : UcViewModelBase
{
public DUT_End_TestViewModel()
{
FileName = "DUT End of Test";
}

}
}
```

DUT_Start_TestViewModel.cs

```
using MT.TestStudio.GUI.ViewModels;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public class DUT_Start_TestViewModel : UcViewModelBase
{
public DUT_Start_TestViewModel()
{
FileName = "DUT Start of Test";
}

}
}
```

DUT_TestViewModel.cs

```
using System.Data;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using Telerik.Windows.Controls;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System;
using System.Collections.Generic;
using MT.TestStudio.GUI.ViewModels;
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Services;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Linq;
using System.Diagnostics;
using MerlinTestStudio_Demo_Telerik.Data.Models;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public interface IDUTTestView
{
void ForceCommitEdit();
/// <summary>
```

```
/// Clears the focus on the GridView respresenting the DUT_Test sequence.
/// </summary>
void ClearGridViewFocus();

/// <summary>
/// Brings a group object into view on the GridView respresenting the DUT_Test sequence.
/// </summary>
/// <param name="obj"></param>
void ScrollObjectIntoView(ISequenceItem obj);

/// <summary>
/// Rebinds the DataBindings in the GridView.
/// </summary>
void RebindGridView();
}

public class DUT_TestViewModel : PVM
{
public static string UserDialogEntry { get; set; }

#region Collections
ObservableCollection<ISequenceItemComponent> _sequenceItemList = new
ObservableCollection<ISequenceItemComponent>();
readonly List<string> _commandStrings = new List<string>()
{
"Execute",
"Skip",
"Repeat",
"Halt"
};

public List<string> CommandStrings
{
get { return _commandStrings; }
}
public ObservableCollection<ISequenceItem> Groups
{
get { return ParentProject.SequenceItems; }
}
public ObservableCollection<ParameterMapTag> TestParamters
{
get { return ParentProject.ParameterMapTags; }
```

```
}
private ObservableCollection<Data.ParameterMapTag> _autoGeneratedParameters = new
ObservableCollection<ParameterMapTag>();
public ObservableCollection<Data.ParameterMapTag> AutoGeneratedParameters
{
get { return _autoGeneratedParameters; }
set { _autoGeneratedParameters = value; OnPropertyChanged("AutoGeneratedParameters"); }
}
public ObservableCollection<PinModel> Pins
{
get { return ParentProject.PinMapDataModel.RfPins; }
}
public ObservableCollection<ISequenceItemComponent> SequenceItemList
{
get { return _sequenceItemList; }
set
{
_sequenceItemList = value;
OnPropertyChanged("SequenceItemList");
}
}

public ObservableCollection<Data.DLL> ComboBoxDLLCollection
{
get { return ParentProject.DLLs; }
}

#endregion

#region Private Members

private object _tempValue;
//public static ISequenceItem GetSelected;
private ISequenceItem _selected;
private UserControl _control;
private object _functionSelected;

public static bool IsUseTestParameterChecked;
public static bool IsUseMeasureCurrentChecked;

private string _previouslySelectedCommand;
```

```
#endregion

#region Public Members

#region ControlVisibilities

public Visibility DPS100_Enabled { get { return CheckIfControlIsEnabled("DPS"); } }
public Visibility RF100_Enabled { get { return CheckIfControlIsEnabled("RF100"); } }
public Visibility RF110_Enabled { get { return CheckIfControlIsEnabled("RF110"); } }
public Visibility RF200_Enabled { get { return CheckIfControlIsEnabled("RF200"); } }
public Visibility RF300_Enabled { get { return CheckIfControlIsEnabled("RF300"); } }
public Visibility HPA100_Enabled { get { return CheckIfControlIsEnabled("HPA100"); } }
public Visibility SCAL100_Enabled { get { return CheckIfControlIsEnabled("SCAL100"); } }
public Visibility PXIOpenATE_Enabled { get { return CheckIfControlIsEnabled("OpenATE"); } }

#endregion


private Data.DLL _comboBoxDLLSelected;
public Data.DLL ComboBoxDLLSelected
{
get { return _comboBoxDLLSelected; }
set
{
if (value != _comboBoxDLLSelected)
{
_comboBoxDLLSelected = value;
OnPropertyChanged("ComboBoxDLLSelected");
}
}
}
private IDUTTestView _view;
public IDUTTestView View
{
get { return _view; }
set
{
_view = value;
OnPropertyChanged("View");

if (value != null)
{
```

```
this.CommitEditBeforeSave += View.ForceCommitEdit;
}
}
}
public object Value
{
get { return _tempValue; }
set
{
if (value != _tempValue)
{
_tempValue = value;
OnPropertyChanged("TempValue");
}
}
}
public ISequenceItem Selected
{
get { return _selected; }
set
{
if (value != null)
{
_selected = value;
OnPropertyChanged("Selected");
//GetSelected = value;

SequenceItemList = null;
//ControlPanelContent = null;
FunctionSelected = null;
}
}
}

//Functionality of SequenceComponent Controls is currently on pause.
public UserControl ControlPanelContent
{
get { return _control; }
set
{
if (value != _control)
{
```

```
_control = value;
OnPropertyChanged("ControlPanelContent");
}
}
}

public object FunctionSelected
{
get { return _functionSelected; }
set
{
if (value != _functionSelected)
{
_functionSelected = value;
OnPropertyChanged("FunctionSelected");

if (value != null)
{
SelectedSequenceComponentChanged();
}
}
}
}
public string PreviouslySelectedCommand
{
get { return _previouslySelectedCommand; }
set { _previouslySelectedCommand = value;
OnPropertyChanged("PreviouslySelectedCommand"); }
}

#endregion

#region Private Methods

private Visibility CheckIfControlIsEnabled(string InstrumentName)
{
//foreach (Data.Instrument I in PVM.ParentProject.Instruments)
//{
//    if (I.InstrumentName == InstrumentName)
//    {
//        return Visibility.Visible;
//    }
```

```
//}
return Visibility.Collapsed;
}

/// <summary>
/// Occurs when ever the selected SequenceComponent in "Function Sequence" changes.
/// </summary>
private void SelectedSequenceComponentChanged()
{
if (FunctionSelected != null)
{
ISequenceItemComponent ai = FunctionSelected as ISequenceItemComponent;
if (ai.ItemType == SequenceItemType.Function)
{
foreach (Parameter p in (ai as Function).FunctionParameters)
ParentProject.TunnelToData(Selected, p, "READ");

}
else if (ai.ItemType == SequenceItemType.DataControl)
{
//Data.DataControl C = ai as Data.DataControl;
//DataStorage._testToolObjectsListSourceTemp = C.Channels;
//ControlPanelContent = C.SequenceControl;
//foreach (Data.TestToolBusinessObject TT in C.Channels)
//    GroupCheck("READ", Value, TT.Pin, TT.Enableau, TT);

}
else { }
}
}

public string PinSearch(string Channel)
{
if (ParentProject.PinMapDataModel.RfPins != null)
{
foreach (Data.PinModel P in ParentProject.PinMapDataModel.RfPins)
{
if (P.mappings != null)
{
foreach (Data.MappingModel M in P.mappings)
{
if (M.IO == Channel)
```

```
{
return P.PinName;
}
}
}
}
}
return "Not Mapped";
}
//private void GroupCheck(string ReadOrWrite, object PassTempValue, string passPinMapping,
bool ForEnableChecked, Data.TestToolBusinessObject TTBO)
//{
//    bool isGroup = GetSelected is IGroupable;
//
//    if (isGroup != true)
//    {
//        TestParameterValuesInOrOut(1, passPinMapping, ReadOrWrite, PassTempValue,
ForEnableChecked, TTBO);
//    }
//    if (isGroup == true)
//    {
//        TestParameterValuesInOrOut(4, passPinMapping, ReadOrWrite, PassTempValue,
ForEnableChecked, TTBO);
//    }
//}
//private void TestParameterValuesInOrOut(int groupColumn, string PinMapping, string
ReadOrWrite, object TempValue, bool ForEnableChecked, Data.TestToolBusinessObject TTBO)
//{
//    if (GetSelected != null)
//    {
//        string RowReference = GetSelected.ReferenceName;
//
//        foreach (DataRow conRow in DataStorage.m_CSV_Conditions.Rows)
//        {
//            if (RowReference == conRow[groupColumn].ToString())
//            {
//                foreach (DataColumn conCol in DataStorage.m_CSV_Conditions.Columns)
//                {
//                    switch (ReadOrWrite)
//                    {
//                        case "WRITE":
//                            if (ForEnableChecked == true)
```

```
//                        {
//                            if (PinMapping == conCol.ColumnName)
//                            {
//                                conRow[DataStorage.m_CSV_Conditions.Columns.IndexOf(conCol)] =
TempValue;
//                            }
//                        }
//                        break;
//                    case "READ":
//                        if (ForEnableChecked == true)
//                        {
//                            if (PinMapping == conCol.ColumnName)
//                            {
//                                //Data.Control C = FunctionSelected as Data.Control;
//                                TTBO.Voltage =
conRow[DataStorage.m_CSV_Conditions.Columns.IndexOf(conCol)].ToString();
//                            }
//                        }
//                        break;
//                    default:
//                        break;
//                }
//            }
//        }
//    }
//}

#region SI command/state methods

private bool IsPrevCommandHalt()
{
if (PreviouslySelectedCommand == "Halt")
{
EnabledCommandsFromHaltSwitch();
//Selected = null;
return true;
}
else { return false; }
}

private void EnabledCommandsFromHaltSwitch()
```

```
{
Selected.State = "Normal";
foreach (ISequenceItem group in ParentProject.SequenceItems)
{
if (group.Sequence > Selected.Sequence)
{
group.State = "Enabled";
if (group.Command == "Halt")
{
group.State = "Halted";
return;
}
}
}
}

#endregion



#endregion


#region Constructor
public event Action CommitEditBeforeSave;
public event Action ChangeOccured;
public override void PaneClose()
{
this.View = null;
}

public event Action<string> OnChangesMade = (UserControl) => { };

private string _dataFilePath;
public override string DataFilePath
{
get { return _dataFilePath; }
set
{
_dataFilePath = value;
OnPropertyChanged("DataFilePath");
//Header = Path.GetFileName(value);
```

```
}
}

public void ForceOnChangesMade()
{
OnChangesMade("DUT Test");
}

public DUT_TestViewModel(Type contentType, string dataFilePath, MerlinProject parentProject,
ProjectFolder parentFolder) : base(contentType, dataFilePath, parentProject, parentFolder)
{
//// Don't execute the viewmodel code if we're just looking at the XAML in the designer.
//if (DesignerProperties.GetIsInDesignMode(new DependencyObject())) return;

this.DataFilePath = dataFilePath;

this.LoadData();

OnChangesMade += (UserControl) =>
{
ParentProject.ReorderSequenceNumbers();
CallRebindGridView();

if(UserControl == "Test Parameters")
{
((PVM)ParentProject.ProjectTree[2].FolderItems[0]).IsSaveRequired = true;
}
else
{
IsSaveRequired = true;
}
};

AddControlCommand = new DelegateCommand(AddControl);
TempParamValueChanged = new DelegateCommand(ChangeTempParamMappingValue);
ReadTempParamValue = new DelegateCommand(ReadTempParamMappingValue);
SelectedInComboBoxChangedCommand = new
DelegateCommand(SelectedInComboBoxChanged);
UpOrDownCommand = new DelegateCommand(UpOrDown);
MoveSelectedToCommand = new DelegateCommand(MoveSelectedTo);

this.ChangeOccured += ChangeOccuredSubscriber;
```

```
}

public override void SaveData()
{
CommitEditBeforeSave?.Invoke();

ParentProject.SaveDLLsData(ParentProject.DllReferencesFilePath);

ParentProject.ReorderSequenceNumbers();

ParentProject.SaveSequenceItemsData(ParentProject.TestSequencesFilePath);

Console.WriteLine(string.Format("Saved: {0}", this.DataFilePath));

base.SaveData();
}

public override void LoadData()
{
MerlinProject.LoadDLLsData(ParentProject.DllReferencesFilePath, ParentProject);
MerlinProject.LoadSequenceItemsData(ParentProject.TestSequencesFilePath, ParentProject);

base.LoadData();
}

public override void RelinkData()
{
ParentProject.RelinkInternalData();
base.RelinkData();
}

public override void ValidateData()
{
base.ValidateData();
}

private void ChangeOccuredSubscriber() { IsSaveRequired = true; }
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
#endregion
```

```
#region Commands

//Outdated.
# region Add Control Command

public ICommand AddControlCommand { get; set; }

private void AddControl(object commandSender)
{
if (Selected != null)
{
//string ControlName = commandSender.ToString();
//Data.DataControl C = new Data.DataControl() { ControlName = ControlName, SequenceControl
= new UserControls.Tools.TestTool() };
//SequenceItemList.Add(C);
//OnChangesMade("DUT Test");
}
}

#endregion


#region TempParamValueChanged Command

public ICommand TempParamValueChanged { get; set; }

private void ChangeTempParamMappingValue(object obj)
{
if (obj != null && Selected.ItemType != SequenceItemType.Block)
{
var parameter = obj as Parameter;

//var timer = new Stopwatch();

//timer.Start();
ParentProject.WriteToTestParameters(Selected, parameter);
//timer.Stop();

//TimeSpan timeTaken = timer.Elapsed;
//string foo = "Time taken: " + timeTaken.ToString(@"m\:ss\.fff");
//Console.WriteLine(foo);
```

```
OnChangesMade("Test Parameters");
}
}

#endregion

#region ReadTempParamValue Command

public ICommand ReadTempParamValue { get; set; }

private void ReadTempParamMappingValue(object obj)
{
if (obj != null && Selected.ItemType != SequenceItemType.Block)
{
var parameter = obj as Parameter;
ParentProject.TunnelToData(Selected, parameter, "READ");
}
}

#endregion


#region Up/Down Command

public ICommand UpOrDownCommand { get; set; }

/// <summary>
/// Moves the selected Group Up or Down by 1.
/// </summary>
/// <param name="obj"></param>
private void UpOrDown(object obj)
{
if (Selected != null && obj != null)
{
int currentIndex = ParentProject.SequenceItems.IndexOf(Selected);
int newIndex = obj.ToString() == "Up" ? currentIndex - 1 :
obj.ToString() == "Down" ? currentIndex + 1 : currentIndex; //Default uses the current index.

if (newIndex < 0 || newIndex > ParentProject.SequenceItems.Count - 1) //If the new index for row
insertion is out of bounds, return.
return;
```

```
ParentProject.SequenceItems.Move(currentIndex, newIndex);
OnChangesMade("DUT Test");
}
}

#endregion

#region MoveSelectedToCommand

public ICommand MoveSelectedToCommand { get; set; }

private void MoveSelectedTo(object obj)
{
if (Selected != null && obj != null)
{
int oldIndex = ParentProject.SequenceItems.IndexOf(Selected);
int newIndex = int.Parse(obj.ToString()) - 1;

ParentProject.SequenceItems.Move(oldIndex, newIndex);
OnChangesMade("DUT Test");
}
}

#endregion


#region Selected In ComboBox Changed Command

public ICommand SelectedInComboBoxChangedCommand { get; set; }

private void SelectedInComboBoxChanged(object obj)
{
if (obj != null)
{
string ComboBoxValue = obj as string;

switch (ComboBoxValue)
{
case "Execute":
if (IsPrevCommandHalt() == false)
{
```

```
//Code Here
}
break;
case "Skip":
if (IsPrevCommandHalt() == false)
{
//Code Here
}
break;
case "Repeat":
if (IsPrevCommandHalt() == false)
{
//Code Here
}
break;
case "Halt":
Selected.State = "Halted";
foreach (ISequenceItem group in ParentProject.SequenceItems)
if (group.Sequence > Selected.Sequence)
group.State = "Disabled";
//Selected = null;
break;
default:
//SelectedActionChanged(ComboBoxValue);
break;
}
OnChangesMade("DUT Test");
}
}

#endregion

#region Add Sequence Block Command

public ICommand AddBlockCommand { get { return new RelayCommand(AddBlock); } }

private void AddBlock()
{
var block = new SequenceBlock() { ItemName = "Block_1", Command = "Execute" };
if (Selected != null)
{
int InsertIndex = ParentProject.SequenceItems.IndexOf(Selected);
```

```
ParentProject.SequenceItems.Insert(InsertIndex, block);
}
else
{
ParentProject.SequenceItems.Insert(0, block);
}
OnChangesMade("DUT Test");
}

#endregion

#region Import .dll command

public ICommand ImportDLLCommand { get { return new RelayCommand(ImportDLL); } }

/// <summary>
/// Imports a selectable DLL upoon click of ImportDLL Button.
/// </summary>
/// <param name="commandSender"></param>
private void ImportDLL()
{
// Create OpenFileDialog
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();
dlg.DefaultExt = ".dll"; // Default file extension
dlg.Filter = "Application Extension (.dll)|*.dll"; // Filter files by extension
dlg.Title = "Import Extension";

// Launch OpenFileDialog by calling ShowDialog method
Nullable<bool> result = dlg.ShowDialog();

if (result == true)
{
var DLL_Types = MerlinProject.ImportAssmblyTypes(dlg.FileName);
foreach (Data.DLL dll in DLL_Types) { ParentProject.DLLs.Add(dll); }

if (DLL_Types.Count > 0)
{
ComboBoxDLLSelected = DLL_Types[0]; //Selects the first item of a newly imported group of dll
types.
//DLLSelected(DLL_Types[0]);
}
```

```
OnChangesMade("DUT Test");
}
}

#endregion

#region Refresh parameter values command

public ICommand RefreshParameterValuesCommand { get { return new
RelayCommand(RefreshParameterValues); } }

/// <summary>
/// Refreshes the parameter values of the selected function.
/// </summary>
private void RefreshParameterValues()
{
if (FunctionSelected != null)
if ((FunctionSelected as ISequenceItemComponent).ItemType == SequenceItemType.Function)
foreach (Parameter p in (FunctionSelected as Function).FunctionParameters)
ParentProject.TunnelToData(Selected, p, "READ");
}

#endregion

#region RemoveFunction Command

public ICommand RemoveFunctionCommand { get { return new
RelayCommand(RemoveFunction); } }

/// <summary>
/// Removes the selected function from the selected SequenceItem.
/// </summary>
private void RemoveFunction()
{
try
{
if (FunctionSelected != null && Selected != null)
{
//MessageBoxResult messageBoxResult = System.Windows.MessageBox.Show("Are you sure?",
"Remove Function", System.Windows.MessageBoxButton.YesNo);
//if (messageBoxResult == MessageBoxResult.Yes)
//   Selected.SequenceComponents.Remove(FunctionSelected as Function);
```

```
Selected.SequenceComponents.Remove(FunctionSelected as Function);
}
}
catch(Exception ex) { MessageBox.Show(ex.Message); }
}

#endregion


#region AutoGen Show Command

public ICommand OpenAutoMapCommand { get { return new
RelayCommand(AutoGenParamMaps); } }

private void AutoGenParamMaps()
{
if (FunctionSelected != null)
{
RadWindow AutoGen = new RadWindow()
{
Header = "Create Missing Parameters",
Height = 450,
Width = 350,
WindowStartupLocation = System.Windows.WindowStartupLocation.CenterOwner,
Content = new UserControls.Tools.AutoGeneratedChecker() { DataContext = this }
};

var ai = FunctionSelected as ISequenceItemComponent;
if (ai.ItemType == SequenceItemType.Function)
{
var testParamMapTags = ParentProject.ParameterMapTags.ToList();

foreach (Data.Parameter param in (ai as Function).FunctionParameters)
{
var ParamMapTag = new ParameterMapTag()
{
ColumnIndex = -1,
Unit = "",
AttributeType = TestAttributeType.Parameter,
ColumnName = param.ParamName,
IsAutoGen = true
};
```

```
int index = testParamMapTags.FindIndex(p => p.ColumnName == param.ParamName);
if (index < 0)
{
if (param.IsMapped)
{
ParamMapTag.IsAutoGen = false;
AutoGeneratedParameters.Add(ParamMapTag);
}
else { AutoGeneratedParameters.Add(ParamMapTag); }
}
}
}
AutoGen.Show();
}
}

#endregion

#region AutoGen Cancel Command

public ICommand CancelBtnCommand { get { return new DelegateCommand(CancelBtn); } }

private void CancelBtn(object obj)
{
var window = obj as RadWindow;
window.Close();

AutoGeneratedParameters.Clear();
}

#endregion

#region AutoGen Done Command

public ICommand DoneBtnCommand { get { return new DelegateCommand(DoneBtn); } }

private void DoneBtn(object obj)
{
var seqItem = FunctionSelected as ISequenceItemComponent;
var funcParams = (seqItem as Data.Function).FunctionParameters.ToList();
foreach (ParameterMapTag pmt in AutoGeneratedParameters)
```

```
{
int index = funcParams.FindIndex(p => p.ParamName == pmt.ColumnName);
if (index >= 0)
{
var x = funcParams[index];
if (pmt.IsAutoGen)
{
//Adds the TestParameter wrapper object to each test for value binding.
foreach (ITest t in ParentProject.Tests)
t.Parameters.Add(new TestParameter());

pmt.ColumnIndex = ParentProject.ParameterMapTags.Count;
ParentProject.AddColumn(pmt.ColumnName, typeof(string), pmt);
x.ParamMapping = pmt;
}
}
}

var window = obj as RadWindow;
window.Close();

AutoGeneratedParameters.Clear();

//TEMP
//((PVM)ParentProject.ProjectTree[2].FolderItems[0]).IsSaveRequired = true;
OnChangesMade("Test Parameters");
}

#endregion



#region Resync Command

public ICommand CallRebindCommand { get { return new RelayCommand(CallRebindGridView); }
}

/// <summary>
/// Calls the Rebind method built-in to the GridView through the code behind.
/// </summary>
private void CallRebindGridView()
{
View.RebindGridView();
```

```
}

#endregion

//NEW
#region Compile Sequence Command
public ICommand CompileSequenceCommand { get { return new
DelegateCommand(CompileSequence); } }
private void CompileSequence(object obj)
{
Microsoft.Win32.SaveFileDialog saveFileDlg = new Microsoft.Win32.SaveFileDialog()
{ Title = "Save Sequence", FileName = "Sequence", Filter = "XML (*.xml)|*.xml" };
Nullable<bool> result = saveFileDlg.ShowDialog();

if (result == true)
{
MerlinProject.CompileTestSequence(saveFileDlg.FileName, this.ParentProject);

Console.WriteLine($"Compiled Sequence: {saveFileDlg.FileName}");
}
else { return; }
}

#endregion

#endregion

//Move Main functionality to interface soon.
#region GridView ContextMenu Commands



#region Add Test To Group Command

public ICommand AddTestToGroupCommand { get { return new
DelegateCommand(AddTestToGroup); } }

/// <summary>
/// Adds an UnGroupedTest into a GroupedTest's list of tests.
/// </summary>
private void AddTestToGroup(object obj)
{
if (obj != null)
```

```
{
try
{
ISequenceItem seqItem = obj as ISequenceItem;
RadWindow.Prompt("Group Name?", this.OnClosed, "");
if (UserDialogEntry == null) return;

//int seqNum = Selected.Sequence;
ParentProject.AddGroupAttribute(seqItem, UserDialogEntry);

//View.ScrollObjectIntoView(PVM.ParentProject.SequenceItems[seqNum]);

OnChangesMade("DUT Test");
CallRebindGridView();
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
}

#endregion

#region Group Matching Command

public ICommand GroupMatchingCommand { get { return new
DelegateCommand(GroupMatching); } }

private void GroupMatching(object obj)
{
if (obj != null)
{
try
{
ISequenceItem seqItem = obj as ISequenceItem;
RadWindow.Prompt("Group Name?", this.OnClosed, "");
if (UserDialogEntry == null) return;

//int seqNum = Selected.Sequence;
int addedTest = ParentProject.GroupMatching(seqItem as UnGroupedTest, UserDialogEntry);
//assumes UnGroupedTest.
if (addedTest > 0)
{
MessageBox.Show($"{addedTest} Test(s) were added to group {UserDialogEntry}.");
```

```
OnChangesMade("DUT Test");
}
else { MessageBox.Show("The test provided does not have any matches."); }
//View.ScrollObjectIntoView(PVM.ParentProject.SequenceItems[seqNum]);

//CallRebindGridView();
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

}

#endregion

#region Break Group Command

public ICommand BreakGroupCommand { get { return new DelegateCommand(BreakGroup); } }

private void BreakGroup(object obj)
{
if (obj != null)
{
try
{
ISequenceItem seqItem = obj as ISequenceItem;
int seqNum = seqItem.Sequence; //Stores sequence # before the group is dissolved.

ParentProject.DissolveGroup(seqItem as TestGroup); //Assumes TestGroup.

View.ScrollObjectIntoView(ParentProject.SequenceItems[seqNum]);

OnChangesMade("DUT Test");
OnChangesMade("Test Parameters");
//CallRebindGridView();
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
}

#endregion

#region Skip Remaining Command
```

```
public ICommand SkipRemainingCommand { get { return new
DelegateCommand(SkipRemaining); } }

private void SkipRemaining(object obj)
{
if (obj != null)
{
try
{
ISequenceItem seqItem = obj as ISequenceItem;
int index = ParentProject.SequenceItems.IndexOf(seqItem);

for (int i = index; i < ParentProject.SequenceItems.Count; i++)
ParentProject.SequenceItems[i].Command = "Skip";

View.ScrollObjectIntoView(ParentProject.SequenceItems[index]);

OnChangesMade("DUT Test");
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
}

#endregion


#endregion

public void OnClosed(object sender, WindowClosedEventArgs e)
{
UserDialogEntry = null;
if (e.PromptResult != null)
{
UserDialogEntry = e.PromptResult.ToString();
}
}
}

}
```

SessionLoadViewModel.cs

```
using MT.TestStudio.GUI.ViewModels;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public class SessionLoadViewModel : UcViewModelBase
{
public SessionLoadViewModel()
{
FileName = "Session Load";
}

}
}
```

SessionUnloadViewModel.cs

```
using MT.TestStudio.GUI.ViewModels;

namespace MerlinTestStudio_Demo_Telerik.ViewModels
{
public class SessionUnloadViewModel : UcViewModelBase
{
public SessionUnloadViewModel()
{
FileName = "Session Unload";
}

}
}
```

CalDefData.cs

```
using MT.TestStudio.Calibration;
using MT.TestStudio.Enums;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace MT.TestStudio.Common
{
public class CalDefDataV3 : CalDefDataV4
{
private SigGenV3 sourceGenerator;

/// <summary>
/// Default Constructor for the CalDefDataV3V4 class.
/// </summary>
public CalDefDataV3()
{
// Do nothing.
}


/// <summary>
/// Copy Constructor for the CalDefDataV3 class.
/// </summary>
/// <param name="ItemToCopy">The object to copy.</param>
public CalDefDataV3(CalDefDataV3 ItemToCopy)
{
this.SourceGenerator = ItemToCopy.SourceGenerator;
this.SourcePort = ItemToCopy.SourcePort;
this.SourcePortAlias = ItemToCopy.SourcePortAlias;
this.MeasurePort = ItemToCopy.MeasurePort;
this.MeasurePortAlias = ItemToCopy.MeasurePortAlias;
this.Frequency = ItemToCopy.Frequency;
this.Level = ItemToCopy.Level;
this.MeasAtten = ItemToCopy.MeasAtten;
this.MeasurementFilter = ItemToCopy.MeasurementFilter;
this.Modulationtype = ItemToCopy.Modulationtype;
this.Waveform = ItemToCopy.Waveform;
this.DutyCycle = ItemToCopy.DutyCycle;
this.Comment = ItemToCopy.Comment;
}

public new SigGenV3 SourceGenerator
{
get
{
return sourceGenerator;
```

```
}
set
{
sourceGenerator = value;
OnPropertyChanged("SourceGenerator");
}
}
}

public class CalDefDataV4 : INotifyPropertyChanged
{
#region Private Member Variables

private SigGen sourceGenerator { get; set; }
private SrcPort sourcePort { get; set; }
private string sourcePortAlias { get; set; }
private MeasPort measurePort { get; set; }
private string measurePortAlias { get; set; }
private double frequency { get; set; }
private double level { get; set; }
private double measAtten { get; set; }
private MeasFilt measurementFilter { get; set; }
private string modulationtype { get; set; }
private string waveform { get; set; }
private double dutyCycle { get; set; }
private string comment { get; set; }
private short marker { get; set; }
private short analysisSlot { get; set; }

#endregion Private Member Variables

/// <summary>
/// Default Constructor for the CalDefDataV3V4 class.
/// </summary>
public CalDefDataV4()
{
// Do nothing.
}



/// <summary>
/// Copy Constructor for the CalDefDataV3V4 class.
```

```
/// </summary>
/// <param name="ItemToCopy">The object to copy.</param>
public CalDefDataV4(CalDefDataV4 ItemToCopy)
{
this.SourceGenerator = ItemToCopy.SourceGenerator;
this.SourcePort = ItemToCopy.SourcePort;
this.SourcePortAlias = ItemToCopy.SourcePortAlias;
this.MeasurePort = ItemToCopy.MeasurePort;
this.MeasurePortAlias = ItemToCopy.MeasurePortAlias;
this.Frequency = ItemToCopy.Frequency;
this.Level = ItemToCopy.Level;
this.MeasAtten = ItemToCopy.MeasAtten;
this.MeasurementFilter = ItemToCopy.MeasurementFilter;
this.Modulationtype = ItemToCopy.Modulationtype;
this.Waveform = ItemToCopy.Waveform;
this.DutyCycle = ItemToCopy.DutyCycle;
this.Comment = ItemToCopy.Comment;
}

public SigGen SourceGenerator
{
get
{
return sourceGenerator;
}
set
{
sourceGenerator = value;
OnPropertyChanged("SourceGenerator");
}
}

public SrcPort SourcePort
{
get
{
return sourcePort;
}
set
{
sourcePort = value;
OnPropertyChanged("SourcePort");
```

```
}
}

public string SourcePortAlias
{
get
{
return sourcePortAlias;
}
set
{
sourcePortAlias = value;
OnPropertyChanged("SourcePortAlias");
}
}

public MeasPort MeasurePort
{
get
{
return measurePort;
}
set
{
measurePort = value;
OnPropertyChanged("MeasurePort");
}
}

public string MeasurePortAlias
{
get
{
return measurePortAlias;
}
set
{
measurePortAlias = value;
OnPropertyChanged("MeasurePortAlias");
}
}
```

```
public double Frequency
{
get
{
return frequency;
}
set
{
frequency = value;
OnPropertyChanged("Frequency");
}
}

public double Level
{
get
{
return level;
}
set
{
level = value;
OnPropertyChanged("Level");
}
}

public double MeasAtten
{
get
{
return measAtten;
}
set
{
measAtten = value;
OnPropertyChanged("MeasAtten");
}
}

public MeasFilt MeasurementFilter
{
get
```

```
{
return measurementFilter;
}
set
{
measurementFilter = value;
OnPropertyChanged("MeasurementFilter");
}
}

//public ModulationType Modulationtype { get; set; }
public string Modulationtype
{
get
{
return modulationtype;
}
set
{
modulationtype = value;
OnPropertyChanged("Modulationtype");
}
}

public string Waveform
{
get
{
return waveform;
}
set
{
waveform = value;
OnPropertyChanged("Waveform");
}
}

public double DutyCycle
{
get
{
return dutyCycle;
```

```
}
set
{
dutyCycle = value;
OnPropertyChanged("DutyCycle");
}
}

public string Comment
{
get
{
return comment;
}
set
{
comment = value;
OnPropertyChanged("Comment");
}
}

public short Marker
{
get
{
return marker;
}
set
{
marker = value;
OnPropertyChanged("Marker");
}
}

public short AnalysisSlot
{
get
{
return analysisSlot;
}
set
{
```

```
analysisSlot = value;
OnPropertyChanged("AnalysisSlot");
}
}


#region Public Events

/// <summary>
/// The property changed event handler that will be called when a property is updated.
/// </summary>
public event PropertyChangedEventHandler PropertyChanged;

#endregion Public Events

/// <summary>
/// Create the OnPropertyChanged method to raise the event.
/// </summary>
/// <param name="name">Name of the property that's just been updated.</param>
public void OnPropertyChanged(string name)
{
PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null)
{
handler(this, new PropertyChangedEventArgs(name));
}
}

}

public class CalDefDataV5 : INotifyPropertyChanged
{
#region Private Member Variables

private SigGen sourceGenerator { get; set; }
private SrcPort sourcePort { get; set; }
private string sourcePortAlias { get; set; }
private MeasPort measurePort { get; set; }
private string measurePortAlias { get; set; }
private double frequency { get; set; }
private double level { get; set; }
```

```
private double measAtten { get; set; }
private MeasFilt measurementFilter { get; set; }
private string modulationtype { get; set; }
private string waveform { get; set; }
private double dutyCycle { get; set; }
private string comment { get; set; }
private short marker { get; set; }
private short analysisSlot { get; set; }
private double? downconverterIFFreq;
private double? downconverterRFAtten;
private double? downconverterIFAtten;
private bool? downconverterPreamp;
private string externalType;

#endregion Private Member Variables

/// <summary>
/// Default Constructor for the CalDefDataV3V4 class.
/// </summary>
public CalDefDataV5()
{
// Do nothing.
}



/// <summary>
/// Copy Constructor for the CalDefDataV3V4 class.
/// </summary>
/// <param name="ItemToCopy">The object to copy.</param>
public CalDefDataV5(CalDefDataV5 ItemToCopy)
{
this.SourceGenerator = ItemToCopy.SourceGenerator;
this.SourcePort = ItemToCopy.SourcePort;
this.SourcePortAlias = ItemToCopy.SourcePortAlias;
this.MeasurePort = ItemToCopy.MeasurePort;
this.MeasurePortAlias = ItemToCopy.MeasurePortAlias;
this.Frequency = ItemToCopy.Frequency;
this.Level = ItemToCopy.Level;
this.MeasAtten = ItemToCopy.MeasAtten;
this.MeasurementFilter = ItemToCopy.MeasurementFilter;
this.Modulationtype = ItemToCopy.Modulationtype;
this.Waveform = ItemToCopy.Waveform;
```

```
this.DutyCycle = ItemToCopy.DutyCycle;
this.Comment = ItemToCopy.Comment;
this.DownconverterIFAtten = ItemToCopy.DownconverterIFAtten;
this.DownconverterIFFreq = ItemToCopy.DownconverterIFFreq;
this.DownconverterRFAtten = ItemToCopy.DownconverterRFAtten;
this.DownconverterPreamp = ItemToCopy.DownconverterPreamp;
this.ExternalType = ItemToCopy.ExternalType;
}

public SigGen SourceGenerator
{
get
{
return sourceGenerator;
}
set
{
sourceGenerator = value;
OnPropertyChanged("SourceGenerator");
}
}

public SrcPort SourcePort
{
get
{
return sourcePort;
}
set
{
sourcePort = value;
OnPropertyChanged("SourcePort");
}
}

public string SourcePortAlias
{
get
{
return sourcePortAlias;
}
set
```

```
{
sourcePortAlias = value;
OnPropertyChanged("SourcePortAlias");
}
}

public MeasPort MeasurePort
{
get
{
return measurePort;
}
set
{
measurePort = value;
OnPropertyChanged("MeasurePort");
}
}

public string MeasurePortAlias
{
get
{
return measurePortAlias;
}
set
{
measurePortAlias = value;
OnPropertyChanged("MeasurePortAlias");
}
}

public double Frequency
{
get
{
return frequency;
}
set
{
frequency = value;
OnPropertyChanged("Frequency");
```

```
}
}

public double Level
{
get
{
return level;
}
set
{
level = value;
OnPropertyChanged("Level");
}
}

public double MeasAtten
{
get
{
return measAtten;
}
set
{
measAtten = value;
OnPropertyChanged("MeasAtten");
}
}

public MeasFilt MeasurementFilter
{
get
{
return measurementFilter;
}
set
{
measurementFilter = value;
OnPropertyChanged("MeasurementFilter");
}
}
```

```
//public ModulationType Modulationtype { get; set; }
public string Modulationtype
{
get
{
return modulationtype;
}
set
{
modulationtype = value;
OnPropertyChanged("Modulationtype");
}
}

public string Waveform
{
get
{
return waveform;
}
set
{
waveform = value;
OnPropertyChanged("Waveform");
}
}

public short Marker
{
get
{
return marker;
}
set
{
marker = value;
OnPropertyChanged("Marker");
}
}

public short AnalysisSlot
{
```

```
get
{
return analysisSlot;
}
set
{
analysisSlot = value;
OnPropertyChanged("AnalysisSlot");
}
}

public double DutyCycle
{
get
{
return dutyCycle;
}
set
{
dutyCycle = value;
OnPropertyChanged("DutyCycle");
}
}

public string Comment
{
get
{
return comment;
}
set
{
comment = value;
OnPropertyChanged("Comment");
}
}



public double? DownconverterIFFreq
{
get
```

```
{
return downconverterIFFreq;
}
set
{
downconverterIFFreq = value;
OnPropertyChanged("DownconverterIFFreq");
}
}

public double? DownconverterRFAtten
{
get
{
return downconverterRFAtten;
}
set
{
downconverterRFAtten = value;
OnPropertyChanged("DownconverterRFAtten");
}
}
public double? DownconverterIFAtten
{
get
{
return downconverterIFAtten;
}
set
{
downconverterIFAtten = value;
OnPropertyChanged("DownconverterIFAtten");
}
}


public bool? DownconverterPreamp
{
get
{
return downconverterPreamp;
}
```

```
set
{
downconverterPreamp = value;
OnPropertyChanged("DownconverterPreamp");
}
}

public string ExternalType
{
get
{
return externalType;
}
set
{
externalType = value;
OnPropertyChanged("ExternalType");
}
}


#region Public Events

/// <summary>
/// The property changed event handler that will be called when a property is updated.
/// </summary>
public event PropertyChangedEventHandler PropertyChanged;

#endregion Public Events

/// <summary>
/// Create the OnPropertyChanged method to raise the event.
/// </summary>
/// <param name="name">Name of the property that's just been updated.</param>
public void OnPropertyChanged(string name)
{
PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null)
{
handler(this, new PropertyChangedEventArgs(name));
}
```

```
}

}

}
```

CalibrationDefModel.cs

```
using MT.TestStudio.Calibration;
using MT.TestStudio.Common;
using MT.TestStudio.Enums;
using MT.TestStudio.Exceptions;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using System.Xml.Linq;
using System.Xml.Schema;

namespace MerlinTest.Tools.TestStudio.Model
{
/// <summary>
/// Class representing the MODEL object of the Calibration Definition View-ViewModel
/// </summary>
//public class CalibrationDefModel
//{
//    #region Private Member Variables

//    #endregion Private Member Variables

//    #region Constructor

//    /// <summary>
//    /// Constructor for the Calibration Definition model.
//    /// </summary>
//    public CalibrationDefModel()
//    {
```

```
//        this.CalVersion = 1;

//        //// Instantiate CalDataV3 and set some defaults.
//        //Info infoV3 = new Info();
//        //CalSettings CalSelectV3 = new CalSettings();
//        //ExternalAttenuation attenExternalV3 = new ExternalAttenuation();
//        //List<CalDataVer3> caldatumV3 = new List<CalDataVer3>();

//        //CalDataV3 = Tuple.Create(caldatumV3, infoV3, CalSelectV3, attenExternalV3);

//        //Info infoV4 = new Info();
//        //CalSettings CalSelectV4 = new CalSettings();
//        //ExternalAttenuation attenExternalV4 = new ExternalAttenuation();
//        //List<CalDataPortModule> calDatumV4 = new List<CalDataPortModule>();



//        //CalDataV4 = Tuple.Create(infoV4, CalSelectV4, attenExternalV4, calDatumV4);

//        //Info infoV5 = new Info();
//        //CalSettings CalSelectV5 = new CalSettings();
//        //ExternalAttenuation attenExternalV5 = new ExternalAttenuation();
//        //List<CalDataPortModule> calDatumV5 = new List<CalDataPortModule>();

//        //CalDataV5 = Tuple.Create(infoV5, CalSelectV5, attenExternalV5, calDatumV5);
//        //CalDataV5 = new CalDefDataV5();
//    }

//    #endregion Constructor

//    #region Public Properties

//    /// <summary>
//    /// Object to hold all of the V3 Cal Def Data.
//    /// </summary>
//    //public Tuple<List<CalDataVer3>, Info, CalSettings, ExternalAttenuation> CalDataV3;

//    ///// <summary>
//    ///// Object to hold all of the V4 Cal Def Data.
//    ///// </summary>
//    //public Tuple<Info, CalSettings, ExternalAttenuation, List<CalDataPortModule>> CalDataV4;

//    /// <summary>
```

```
//    /// Object to hold all of the V5 Cal Def Data.
//    /// </summary>
//    //public Tuple<Info, CalSettings, ExternalAttenuation, List<CalDataPortModule>> CalDataV5;

//    /// <summary>
//    /// Object to hold all of the V5 Cal Def Data.
//    /// </summary>
//    public List<CalDefDataV5> CalDataV5 { get; set; } = new List<CalDefDataV5>();

//    /// <summary>
//    /// Property to represent the product name that the calibration file is associated with.
//    /// </summary>
//    public string ProductName;

//    /// <summary>
//    /// Revision of the calibration file.
//    /// </summary>
//    public string Revision;

//    /// <summary>
//    /// The test program.
//    /// </summary>
//    public string TestProgram;

//    /// <summary>
//    /// String to indicate which current CAL file has been loaded.
//    /// </summary>
//    public int CalVersion { get; set; }

//    /// <summary>
//    /// Comment associated with the calibration file.
//    /// </summary>
//    public string Comment;

//    /// <summary>
//    /// Property to indicate if RF Source/Measure paths should be calibrated.
//    /// </summary>
//    public bool CalibrateRfSourceMeasure;

//    /// <summary>
//    /// Propert to indicate if the internal path should be calibrated.
//    /// </summary>
```

```
//    public bool CalInternalPath;

//    /// <summary>
//    /// Property to indicate if NOISE SRC could be calibrated.
//    /// </summary>
//    public bool CalNoiseSrc;

//    /// <summary>
//    /// Property to indicate if VNA should be calibrated.
//    /// </summary>
//    public bool CalVNA;

//    /// <summary>
//    /// Property to indicate if the external downconverter should be used.
//    /// </summary>
//    public bool UseExternalDownConverter;

//    /// <summary>
//    /// Gets / Sets Noise bandwidth.
//    /// </summary>
//    public double NoiseBandwidth;

//    /// <summary>
//    /// Array to hold the 17 source attenuation values.
//    /// </summary>
//    public double[] SourceAttenutaion = new double[17];

//    /// <summary>
//    /// Array to hold the 30 measure attenuation values.
//    /// </summary>
//    public double[] MeasureAttenutaion = new double[30];

//    #endregion Public Properties

//    #region Public Events / Delegates

//    /// <summary>
//    /// Event of type PropertyChangedEventHandler which is used to implement
INotifyPropertyChanged
//    /// and provide notification of when a property has changed.
//    /// </summary>
//    public event PropertyChangedEventHandler PropertyChanged;
```

```
//    #endregion Public Events / Delegates

//    #region Public Methods

//    /// <summary>
//    /// Helper method that converts from the Cal Def data file format for a
//    /// measure path being populated to a bool.
//    /// </summary>
//    /// <param name="MeasurePopulated"></param>
//    /// <returns>True if an 'x' was passed as an argument and false otherwise.</returns>
//    private static bool ConvertMeasurePopulatedToBool(string MeasurePopulated)
//    {
//        if (MeasurePopulated == "x" || MeasurePopulated == "x")
//        {
//            return true;
//        }
//        else
//        {
//            return false;
//        }

//    }

//    /// <summary>
//    /// Method that causes a file to be read in.
//    /// </summary>
//    /// <param name="FileName">The file to be read in.</param>
//    public void LoadDataFile(string FileName)
//    {
//        try
//        {
//            // This method is called by the framework to get the schema for this type.
//            // We return an existing schema from disk.
//            Assembly asmb = Assembly.GetExecutingAssembly();
//            string[] names = asmb.GetManifestResourceNames();
//            Stream stream =
asmb.GetManifestResourceStream("MT.TestStudio.Model.Schemas.CalibrationDefinition.xsd");

//            XmlSchema vsvSchema = XmlSchema.Read(stream, null);
//            XmlSchemaSet schemas = new XmlSchemaSet();
```

```
//          schemas.Add(vsvSchema);
//          XDocument document = XDocument.Load(FileName);
//          string msg = "";
//          document.Validate(schemas, (o, e) =>
//          {
//              msg += e.Message + "(Line " + e.Exception.LineNumber + ") " +
Environment.NewLine;
//          });

//          if (msg == string.Empty)
//          {
//              this.CalVersion =
HelperMethods.ConvertStringToIntSafely(document.Root.Attribute("Version").Value);

//              if (this.CalVersion == 1)
//              {

//                  // Work out what the namespace is for the XML file.
//                  XNamespace calDefNS = document.Root.Name.Namespace;

//                  var result = document.Descendants(calDefNS + "Settings").Select(cd => new
//                  {
//                      Product = cd.Element(calDefNS + "Product").Value,
//                      Revision = cd.Element(calDefNS + "Revision").Value,
//                      TestProgram = cd.Element(calDefNS + "TestProgram").Value,
//                      Comment = cd.Element(calDefNS + "Comment").Value,
//                      CalibrateRFSourceMeasure = cd.Element(calDefNS +
"CalibrateRFSourceMeasure").Value,
//                      CalibrateInternalDirectPath = cd.Element(calDefNS +
"CalibrateInternalDirectPath").Value,
//                      CalibrateNoiseSource = cd.Element(calDefNS + "CalibrateNoiseSource").Value,
//                      CalibrateVNA = cd.Element(calDefNS + "CalibrateVNA").Value,
//                      NoiseBandwidth = cd.Element(calDefNS + "NoiseBandwidth").Value,
//                  });

//                  foreach (var cd in result)
//                  {
//                      this.ProductName = cd.Product;
//                      this.Revision = cd.Revision;
//                      this.TestProgram = cd.TestProgram;
//                      this.Comment = cd.Comment;
//                      this.CalibrateRfSourceMeasure = bool.Parse(cd.CalibrateRFSourceMeasure);
```

```
//                  this.CalInternalPath = bool.Parse(cd.CalibrateInternalDirectPath);
//                  this.CalNoiseSrc = bool.Parse(cd.CalibrateNoiseSource);
//                  this.CalVNA = bool.Parse(cd.CalibrateVNA);
//                  this.NoiseBandwidth =
HelperMethods.ConvertStringToDouble(cd.NoiseBandwidth);
//              }

//              List<XElement> sourceAttenXmlList = (from e in document.Descendants(calDefNS
+ "Settings").Descendants(calDefNS + "SourceAttenuation").Elements()

//                                  select e).ToList();

//              foreach(XElement ele in sourceAttenXmlList)
//              {
//                  Console.WriteLine("{0}", ele.Value);
//              }

//              for(int index = 0; index < SourceAttenutaion.Length; index++)
//              {
//                  SourceAttenutaion[index] =
HelperMethods.ConvertStringToDouble(sourceAttenXmlList[index].Value);
//              }

//              List<XElement> measureAttenXmlList = (from e in
document.Descendants(calDefNS + "Settings").Descendants(calDefNS +
"MeasureAttenuation").Elements()
//                                      select e).ToList();

//              for (int index = 0; index < MeasureAttenutaion.Length; index++)
//              {
//                  MeasureAttenutaion[index] =
HelperMethods.ConvertStringToDouble(measureAttenXmlList[index].Value);
//              }

//              List<XElement> calPointXmlList = (from e in document.Descendants(calDefNS +
"Settings").Descendants(calDefNS + "CalPointList")
//                                       select e).ToList();


//              foreach(XElement xe in calPointXmlList)
//              {
//                  CalDefDataV5 tmp = new CalDefDataV5();
```

```
//                  tmp.SourceGenerator =
HelperMethods.GetEnumFromString<SigGen>(xe.Element(calDefNS + "Source").Value);
//                  tmp.SourcePort =
HelperMethods.GetEnumFromString<SrcPort>(xe.Element(calDefNS + "SourcePort").Value);
//                  tmp.MeasurePort =
HelperMethods.GetEnumFromString<MeasPort>(xe.Element(calDefNS + "MeasurePort").Value);
//                  tmp.Frequency = HelperMethods.ConvertStringToDouble(xe.Element(calDefNS +
"Frequency").Value);
//                  tmp.Level = HelperMethods.ConvertStringToDouble(xe.Element(calDefNS +
"Level").Value);
//                  tmp.MeasurementFilter =
HelperMethods.GetEnumFromString<MeasFilt>(xe.Element(calDefNS + "Filter").Value);
//                  tmp.MeasAtten = HelperMethods.ConvertStringToDouble(xe.Element(calDefNS +
"MeasAtten").Value);
//                  tmp.Modulationtype = xe.Element(calDefNS + "ModulationType").Value;
//                  tmp.Waveform = xe.Element(calDefNS + "ModulationFile").Value;
//                  tmp.Marker = HelperMethods.ConvertStringToshort(xe.Element(calDefNS +
"Marker").Value);
//                  tmp.AnalysisSlot = HelperMethods.ConvertStringToshort(xe.Element(calDefNS +
"AnalysisSlot").Value);
//                  tmp.DownconverterIFFreq =
HelperMethods.ConvertStringToDoubleNullable(xe.Element(calDefNS +
"DownconverterIFFreq").Value);
//                  tmp.DownconverterRFAtten =
HelperMethods.ConvertStringToDoubleNullable(xe.Element(calDefNS +
"DownconverterRFAtten").Value);
//                  tmp.DownconverterIFAtten =
HelperMethods.ConvertStringToDoubleNullable(xe.Element(calDefNS +
"DownconverterIFAtten").Value);
//                  tmp.DownconverterPreamp =
HelperMethods.ConvertStringToBoolNullable(xe.Element(calDefNS +
"DownconverterPreamp").Value);
//                  tmp.ExternalType = xe.Element(calDefNS + "ExternalType").Value;
//                  tmp.Comment = xe.Element(calDefNS + "Comment").Value;

//                  CalDataV5.Add(tmp);
//              }
//          }
//          else
//          {
//              throw new CalibrationDefinitionException("Calibration Definition Version of " +
```

```
this.CalVersion + " not supported!");
//              }
//          }
//          else
//          {
//              throw new CalibrationDefinitionException("Document invalid: " + msg);
//          }

//          // Notify the View-Model that the product config data has been read in.
//          OnPropertyChanged("CsvDataLoaded");
//      }
//      catch (Exception ex)
//      {
//          string message = string.Format("An error occured while loading a product configuration
file {0}. {1}", FileName, ex.Message);
//          TestStudioModel.trace.TraceError(message);
//          throw new CalibrationDefinitionException(message);
//      }
//      //CalImport Importer = new CalImport();

//      //// Read header of file and split by ','
//      //var headerInfo = (from lines in File.ReadLines(FileName).Take(5)
//      //                  let columns = lines.Split(',')
//      //                  select columns).ToList();


//      //foreach (string[] strArr in headerInfo)
//      //{
//      //    if (strArr.Length > 1)
//      //    {
//      //        if (strArr[0] == "Calibration Version:")
//      //        {
//      //            this.CalVersion = strArr[1];
//      //        }
//      //    }
//      //}

//      //switch (this.CalVersion)
//      //{
//      //    case "v3":
//      //        {
//      //            CalDataV3 = Importer.ImportCalConfigVer3(FileName);
```

```
//        //            break;
//        //        }
//        //    case "v4":
//        //        {
//        //            CalDataV4 = Importer.ImportCalConfigVer4V2(FileName, false);
//        //            break;
//        //        }
//        //    case "v5":
//        //        {
//        //            CalDataV5 = Importer.ImportCalConfigVer5V2(FileName, false);
//        //            break;
//        //        }
//        //    default:
//        //        throw new CalibrationDefinitionException("Calibration File Version not supported : " +
this.CalVersion);
//        //}

//        // Notify the View-Model that the CSV data has been reloaded.
//        OnPropertyChanged("CsvDataLoaded");

//    }

//    ///// <summary>
//    ///// Method that causes a file to be read in.
//    ///// </summary>
//    ///// <param name="FileName">The file to be read in.</param>
//    //public void LoadCsvDataFile(string FileName)
//    //{
//    //    CalImport Importer = new CalImport();

//    //    // Read header of file and split by ','
//    //    var headerInfo = (from lines in File.ReadLines(FileName).Take(5)
//    //                      let columns = lines.Split(',')
//    //                      select columns).ToList();


//    //   foreach(string[] strArr in headerInfo)
//    //    {
//    //        if ( strArr.Length > 1)
//    //        {
//    //            if ( strArr[0] == "Calibration Version:")
//    //            {
```

```
//    //                this.CalVersion = strArr[1];
//    //            }
//    //        }
//    //    }

//    //   switch(this.CalVersion)
//    //    {
//    //        case "v3":
//    //            {
//    //                CalDataV3 = Importer.ImportCalConfigVer3(FileName);
//    //                break;
//    //            }
//    //        case "v4":
//    //            {
//    //                CalDataV4 = Importer.ImportCalConfigVer4V2(FileName, false);
//    //                break;
//    //            }
//    //        case "v5":
//    //            {
//    //                CalDataV5 = Importer.ImportCalConfigVer5V2(FileName,false);
//    //                break;
//    //            }
//    //        default:
//    //            throw new CalibrationDefinitionException("Calibration File Version not supported : " +
this.CalVersion);
//    //    }

//    //    // Notify the View-Model that the CSV data has been reloaded.
//    //    OnPropertyChanged("CsvDataLoaded");

//    //}

//    /// <summary>
//    /// Helper method to convert bool to the string Y or N.
//    /// </summary>
//    /// <param name="ValueToConvert"></param>
//    /// <returns></returns>
//    private string ConvertBoolToYN(bool ValueToConvert)
//    {
//        if ( ValueToConvert == true )
//        {
//            return "Y";
```

```
//        }
//        else
//        {
//            return "N";
//        }
//    }

//    /// <summary>
//    /// Method that causes the data in the to be written to the file associated with this instance of the Product Configuration viewModel.
//    /// </summary>
//    public void SaveDataFile(string FileName)
//    {
//        try
//        {
//            // Define some settings for the format of the XML file.
//            XmlWriterSettings xmlWriterSettings = new XmlWriterSettings();
//            xmlWriterSettings.NewLineOnAttributes = false;
//            xmlWriterSettings.Indent = true;

//            //System.Security.Cryptography.MACTripleDES hash = new System.Security.Cryptography.MACTripleDES(Encoding.Default.GetBytes("mykey"));
//            //string hashString = Convert.ToBase64String(hash.ComputeHash(Encoding.Default.GetBytes(myXMLString)));

//            using (XmlWriter xmlWriter = XmlWriter.Create(FileName, xmlWriterSettings))
//            {
//                xmlWriter.WriteStartDocument();
//                xmlWriter.WriteStartElement("CalConfig", "http://tempuri.org/XMLSchema1.xsd");
//                xmlWriter.WriteAttributeString("Version", "1");

//                xmlWriter.WriteStartElement("Settings");



//                xmlWriter.WriteStartElement("Revision");
//                xmlWriter.WriteValue(this.Revision);
//                xmlWriter.WriteEndElement(); // Revision

//                xmlWriter.WriteStartElement("TestProgram");
//                xmlWriter.WriteValue(this.TestProgram);
//                xmlWriter.WriteEndElement(); // TestProgram
```

```
//              xmlWriter.WriteStartElement("Comment");
//              xmlWriter.WriteValue(this.Comment);
//              xmlWriter.WriteEndElement(); // Comment

//              xmlWriter.WriteStartElement("CalibrateRFSourceMeasure");
//              xmlWriter.WriteValue(this.CalibrateRfSourceMeasure);
//              xmlWriter.WriteEndElement(); // CalibrateRFSourceMeasure

//              xmlWriter.WriteStartElement("CalibrateInternalDirectPath");
//              xmlWriter.WriteValue(this.CalInternalPath);
//              xmlWriter.WriteEndElement(); // CalibrateInternalDirectPath

//              xmlWriter.WriteStartElement("CalibrateNoiseSource");
//              xmlWriter.WriteValue(this.CalNoiseSrc);
//              xmlWriter.WriteEndElement(); // CalibrateNoiseSource


//              xmlWriter.WriteStartElement("CalibrateVNA");
//              xmlWriter.WriteValue(this.CalVNA);
//              xmlWriter.WriteEndElement(); // CalibrateVNA

//              xmlWriter.WriteStartElement("NoiseBandwidth");
//              xmlWriter.WriteValue(this.NoiseBandwidth);
//              xmlWriter.WriteEndElement(); // NoiseBandwidth

//              #region Source Attenuation

//              xmlWriter.WriteStartElement("SourceAttenuation");

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P1");
//              xmlWriter.WriteValue(this.SourceAttenutaion[0]);
//              xmlWriter.WriteEndElement(); // Attenuation P1

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P2");
//              xmlWriter.WriteValue(this.SourceAttenutaion[1]);
//              xmlWriter.WriteEndElement(); // Attenuation P2

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P3");
//              xmlWriter.WriteValue(this.SourceAttenutaion[2]);
```

```
//              xmlWriter.WriteEndElement(); // Attenuation P3

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P4");
//              xmlWriter.WriteValue(this.SourceAttenutaion[3]);
//              xmlWriter.WriteEndElement(); // Attenuation P4

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P5");
//              xmlWriter.WriteValue(this.SourceAttenutaion[4]);
//              xmlWriter.WriteEndElement(); // Attenuation P5

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P1");
//              xmlWriter.WriteValue(this.SourceAttenutaion[0]);
//              xmlWriter.WriteEndElement(); // Attenuation P1

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P6");
//              xmlWriter.WriteValue(this.SourceAttenutaion[5]);
//              xmlWriter.WriteEndElement(); // Attenuation P6

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P7");
//              xmlWriter.WriteValue(this.SourceAttenutaion[6]);
//              xmlWriter.WriteEndElement(); // Attenuation P7

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P8");
//              xmlWriter.WriteValue(this.SourceAttenutaion[7]);
//              xmlWriter.WriteEndElement(); // Attenuation P8

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P9");
//              xmlWriter.WriteValue(this.SourceAttenutaion[8]);
//              xmlWriter.WriteEndElement(); // Attenuation P9

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P10");
//              xmlWriter.WriteValue(this.SourceAttenutaion[9]);
//              xmlWriter.WriteEndElement(); // Attenuation P10
```

```
//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P11");
//              xmlWriter.WriteValue(this.SourceAttenutaion[10]);
//              xmlWriter.WriteEndElement(); // Attenuation P11

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P12");
//              xmlWriter.WriteValue(this.SourceAttenutaion[11]);
//              xmlWriter.WriteEndElement(); // Attenuation P12

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P13");
//              xmlWriter.WriteValue(this.SourceAttenutaion[12]);
//              xmlWriter.WriteEndElement(); // Attenuation P13

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P14");
//              xmlWriter.WriteValue(this.SourceAttenutaion[13]);
//              xmlWriter.WriteEndElement(); // Attenuation P14

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P15");
//              xmlWriter.WriteValue(this.SourceAttenutaion[14]);
//              xmlWriter.WriteEndElement(); // Attenuation P15

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "P16");
//              xmlWriter.WriteValue(this.SourceAttenutaion[15]);
//              xmlWriter.WriteEndElement(); // Attenuation P16

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "SG1");
//              xmlWriter.WriteValue(this.SourceAttenutaion[16]);
//              xmlWriter.WriteEndElement(); // Attenuation SG1


//              xmlWriter.WriteEndElement(); // SourceAttenuation

//              #endregion Source Attenuation

//              #region MeasureAttenuation
```

```
//              xmlWriter.WriteStartElement("MeasureAttenuation");

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M1");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[0]);
//              xmlWriter.WriteEndElement(); // Attenuation M1

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M2");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[1]);
//              xmlWriter.WriteEndElement(); // Attenuation M2

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M3");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[2]);
//              xmlWriter.WriteEndElement(); // Attenuation M3

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M4");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[3]);
//              xmlWriter.WriteEndElement(); // Attenuation M4

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M5");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[4]);
//              xmlWriter.WriteEndElement(); // Attenuation M5

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M1");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[0]);
//              xmlWriter.WriteEndElement(); // Attenuation M1

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M6");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[5]);
//              xmlWriter.WriteEndElement(); // Attenuation M6

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M7");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[6]);
//              xmlWriter.WriteEndElement(); // Attenuation M7
```

```
//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M8");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[7]);
//              xmlWriter.WriteEndElement(); // Attenuation M8

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M9");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[8]);
//              xmlWriter.WriteEndElement(); // Attenuation M9

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M10");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[9]);
//              xmlWriter.WriteEndElement(); // Attenuation M10

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M11");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[10]);
//              xmlWriter.WriteEndElement(); // Attenuation M11

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M12");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[11]);
//              xmlWriter.WriteEndElement(); // Attenuation M12

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M13");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[12]);
//              xmlWriter.WriteEndElement(); // Attenuation M13

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M14");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[13]);
//              xmlWriter.WriteEndElement(); // Attenuation M14

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M15");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[14]);
//              xmlWriter.WriteEndElement(); // Attenuation M15

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M16");
```

```
//              xmlWriter.WriteValue(this.MeasureAttenutaion[15]);
//              xmlWriter.WriteEndElement(); // Attenuation M16

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M17");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[16]);
//              xmlWriter.WriteEndElement(); // Attenuation M17

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M18");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[17]);
//              xmlWriter.WriteEndElement(); // Attenuation M18

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M19");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[18]);
//              xmlWriter.WriteEndElement(); // Attenuation M19

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M20");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[19]);
//              xmlWriter.WriteEndElement(); // Attenuation M20

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M21");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[20]);
//              xmlWriter.WriteEndElement(); // Attenuation M21

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M22");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[21]);
//              xmlWriter.WriteEndElement(); // Attenuation M22

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M23");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[22]);
//              xmlWriter.WriteEndElement(); // Attenuation M23

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M24");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[23]);
//              xmlWriter.WriteEndElement(); // Attenuation M24
```

```
//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M25");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[24]);
//              xmlWriter.WriteEndElement(); // Attenuation M25

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "M26");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[25]);
//              xmlWriter.WriteEndElement(); // Attenuation M26

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "MA");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[26]);
//              xmlWriter.WriteEndElement(); // Attenuation MA

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "MB");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[27]);
//              xmlWriter.WriteEndElement(); // Attenuation MB

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "MC");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[28]);
//              xmlWriter.WriteEndElement(); // Attenuation MC

//              xmlWriter.WriteStartElement("Attenuation");
//              xmlWriter.WriteAttributeString("id", "MD");
//              xmlWriter.WriteValue(this.MeasureAttenutaion[29]);
//              xmlWriter.WriteEndElement(); // Attenuation MD

//              xmlWriter.WriteEndElement(); // MeasureAttenuation

//              #endregion MeasureAttenuation

//              #region CalPointList

//              for (int index = 0; index < this.CalDataV5.Count; index++)
//              {

//                  xmlWriter.WriteStartElement("CalPointList");
```

```
//                xmlWriter.WriteStartElement("Source");
//                xmlWriter.WriteValue(this.CalDataV5[index].SourceGenerator.ToString());
//                xmlWriter.WriteEndElement(); // Source

//                xmlWriter.WriteStartElement("SourcePort");
//                xmlWriter.WriteValue(this.CalDataV5[index].SourcePort.ToString());
//                xmlWriter.WriteEndElement(); // SourcePort

//                xmlWriter.WriteStartElement("MeasurePort");
//                xmlWriter.WriteValue(this.CalDataV5[index].MeasurePort.ToString());
//                xmlWriter.WriteEndElement(); // MeasurePort

//                xmlWriter.WriteStartElement("Frequency");
//                xmlWriter.WriteValue(this.CalDataV5[index].Frequency.ToString());
//                xmlWriter.WriteEndElement(); // Frequency

//                xmlWriter.WriteStartElement("Level");
//                xmlWriter.WriteValue(this.CalDataV5[index].Level.ToString());
//                xmlWriter.WriteEndElement(); // Level

//                xmlWriter.WriteStartElement("Filter");
//                xmlWriter.WriteValue(this.CalDataV5[index].MeasurementFilter.ToString());
//                xmlWriter.WriteEndElement(); // Filter

//                xmlWriter.WriteStartElement("MeasAtten");
//                xmlWriter.WriteValue(this.CalDataV5[index].MeasAtten.ToString());
//                xmlWriter.WriteEndElement(); // MeasAtten

//                xmlWriter.WriteStartElement("ModulationType");
//
xmlWriter.WriteValue(HelperMethods.ConvertObjectToString(this.CalDataV5[index].Modulationtyp
e));
//                xmlWriter.WriteEndElement(); // ModulationType

//                xmlWriter.WriteStartElement("ModulationFile");
//
xmlWriter.WriteValue(HelperMethods.ConvertObjectToString(this.CalDataV5[index].Waveform));
//                xmlWriter.WriteEndElement(); // ModulationFile

//                xmlWriter.WriteStartElement("Marker");
//                xmlWriter.WriteValue(this.CalDataV5[index].Marker.ToString());
//                xmlWriter.WriteEndElement(); // Marker
```

```
//                xmlWriter.WriteStartElement("AnalysisSlot");
//                xmlWriter.WriteValue(this.CalDataV5[index].AnalysisSlot.ToString());
//                xmlWriter.WriteEndElement(); // AnalysisSlot

//                xmlWriter.WriteStartElement("DownconverterIFFreq");
//                xmlWriter.WriteValue(this.CalDataV5[index].DownconverterIFFreq.ToString());
//                xmlWriter.WriteEndElement(); // DownconverterIFFreq

//                xmlWriter.WriteStartElement("DownconverterRFAtten");
//                xmlWriter.WriteValue(this.CalDataV5[index].DownconverterRFAtten.ToString());
//                xmlWriter.WriteEndElement(); // DownconverterRFAtten

//                xmlWriter.WriteStartElement("DownconverterIFAtten");
//                xmlWriter.WriteValue(this.CalDataV5[index].DownconverterIFAtten.ToString());
//                xmlWriter.WriteEndElement(); // DownconverterIFAtten

//                xmlWriter.WriteStartElement("DownconverterPreamp");
//                xmlWriter.WriteValue(this.CalDataV5[index].DownconverterPreamp.ToString());
//                xmlWriter.WriteEndElement(); // DownconverterPreamp

//                xmlWriter.WriteStartElement("ExternalType");
//
xmlWriter.WriteValue(HelperMethods.ConvertObjectToString(this.CalDataV5[index].ExternalType)
);
//                xmlWriter.WriteEndElement(); // ExternalType

//                xmlWriter.WriteStartElement("Comment");
//
xmlWriter.WriteValue(HelperMethods.ConvertObjectToString(this.CalDataV5[index].Comment));
//                xmlWriter.WriteEndElement(); // Comment

//                xmlWriter.WriteEndElement(); // CalPointList

//            }

//            #endregion CalPointlist

//            xmlWriter.WriteEndElement(); // Settings

//            xmlWriter.WriteEndDocument();
//        }
```

```
//            XmlTextReader r = new XmlTextReader(FileName);
//            while (r.Read())
//            {
//                if (r.NodeType == XmlNodeType.Element &&
//                r.Name == "Settings")
//                    Console.WriteLine(r.ReadOuterXml());
//            }
//            r.Close();

//        }
//        catch (Exception ex)
//        {
//            string message = string.Format("An error occured while saving a product configuration file
{0}. {1}", FileName, ex.Message);
//            TestStudioModel.trace.TraceInformation(message);
//            throw new ProductConfigurationException(message);
//        }
//    }

//    ///// <summary>
//    ///// Method that causes the data in the to be written to the file associated with this instance of
the Product Configuration viewModel.
//    ///// </summary>
//    //public void SaveCsvDataFile(string FileName)
//    //{
//    //    try
//    //    {
//    //        switch (this.CalVersion)
//    //        {
//    //            case "v3":
//    //                {
//    //                    using (StreamWriter sw = new StreamWriter(FileName))
//    //                    {
//    //                        // Info
//    //                        sw.WriteLine("Product:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,",
this.CalDataV3.Item2.Product);
//    //                        sw.WriteLine("Revision:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,",
this.CalDataV3.Item2.Revision);
//    //                        sw.WriteLine("Test Program:,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");
//    //                        sw.WriteLine("Calibration Version:,v3,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");
//    //                        sw.WriteLine("Comment:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,",
```

```
this.CalDataV3.Item2.Comment);
//    //                        sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");
//    //                        // Cal Settings
//    //                        sw.WriteLine("Use External Power Meter:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV3.Item3.PowerMeterAvailable));
//    //                        sw.WriteLine("Calibrate RF Source/Measure:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV3.Item3.SourceMeasurePath));
//    //                        sw.WriteLine("Calibrate Internal Matrix Path:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV3.Item3.InternalMatrixPath));
//    //                        sw.WriteLine("Calibrate Noise Source:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV3.Item3.NoiseSource));
//    //                        sw.WriteLine("Calibrate DC Sense:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV3.Item3.DCSense));
//    //                        sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //                        // External Attenuators
//    //
sw.WriteLine(",P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15,P16,SG1,,,,,,,,,,,,,,,,,,,,,,,
,,,,");
//    //                        sw.WriteLine("Source Path
Attenuators:,{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},{12},{13},{14},{15},{16},,,,,,,,,,,,,,,,,,,,,,,,,",
//    //
this.CalDataV3.Item4.srcAttenPortModule[0],
//    //
this.CalDataV3.Item4.srcAttenPortModule[1],
//    //
this.CalDataV3.Item4.srcAttenPortModule[2],
//    //
this.CalDataV3.Item4.srcAttenPortModule[3],
//    //
this.CalDataV3.Item4.srcAttenPortModule[4],
//    //
this.CalDataV3.Item4.srcAttenPortModule[5],
//    //
this.CalDataV3.Item4.srcAttenPortModule[6],
//    //
this.CalDataV3.Item4.srcAttenPortModule[7],
//    //
this.CalDataV3.Item4.srcAttenPortModule[8],
//    //
this.CalDataV3.Item4.srcAttenPortModule[9],
//    //
```

```
this.CalDataV3.Item4.srcAttenPortModule[10],
//    //
this.CalDataV3.Item4.srcAttenPortModule[11],
//    //
this.CalDataV3.Item4.srcAttenPortModule[12],
//    //
this.CalDataV3.Item4.srcAttenPortModule[13],
//    //
this.CalDataV3.Item4.srcAttenPortModule[14],
//    //
this.CalDataV3.Item4.srcAttenPortModule[15],
//    //
this.CalDataV3.Item4.srcAttenPortModule[16]);

//    //                        sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //
sw.WriteLine(",P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15,P16,M1,M2,M3,M4,M5,
M6,M7,M8,M9,M10,M11,M12,M13,M14,M15,M16,M17,M18,M19,M20,M21,M22,M23,M24,M25,M
26,MA,MB,MC,MD");
//    //                        sw.WriteLine("Measure Path
Attenuators:,{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},{12},{13},{14},{15},{16},{17},{18},{19},{20},
{21},{22},{23},{24},{25},{26},{27},{28},{29},{30},{31},{32},{33},{34},{35},{36},{37},{38},{39},{40},{41},{
42},{43},{44},{45}",
//    //
this.CalDataV3.Item4.measAttenPortModule[0],
//    //
this.CalDataV3.Item4.measAttenPortModule[1],
//    //
this.CalDataV3.Item4.measAttenPortModule[2],
//    //
this.CalDataV3.Item4.measAttenPortModule[3],
//    //
this.CalDataV3.Item4.measAttenPortModule[4],
//    //
this.CalDataV3.Item4.measAttenPortModule[5],
//    //
this.CalDataV3.Item4.measAttenPortModule[6],
//    //
this.CalDataV3.Item4.measAttenPortModule[7],
//    //
this.CalDataV3.Item4.measAttenPortModule[8],
```

```
//    //
this.CalDataV3.Item4.measAttenPortModule[9],
//    //
this.CalDataV3.Item4.measAttenPortModule[10],
//    //
this.CalDataV3.Item4.measAttenPortModule[11],
//    //
this.CalDataV3.Item4.measAttenPortModule[12],
//    //
this.CalDataV3.Item4.measAttenPortModule[13],
//    //
this.CalDataV3.Item4.measAttenPortModule[14],
//    //
this.CalDataV3.Item4.measAttenPortModule[15],
//    //
this.CalDataV3.Item4.measAttenPortModule[16],
//    //
this.CalDataV3.Item4.measAttenPortModule[17],
//    //
this.CalDataV3.Item4.measAttenPortModule[18],
//    //
this.CalDataV3.Item4.measAttenPortModule[19],
//    //
this.CalDataV3.Item4.measAttenPortModule[20],
//    //
this.CalDataV3.Item4.measAttenPortModule[21],
//    //
this.CalDataV3.Item4.measAttenPortModule[22],
//    //
this.CalDataV3.Item4.measAttenPortModule[23],
//    //
this.CalDataV3.Item4.measAttenPortModule[24],
//    //
this.CalDataV3.Item4.measAttenPortModule[25],
//    //
this.CalDataV3.Item4.measAttenPortModule[26],
//    //
this.CalDataV3.Item4.measAttenPortModule[27],
//    //
this.CalDataV3.Item4.measAttenPortModule[28],
//    //
this.CalDataV3.Item4.measAttenPortModule[29],
```

```
//    //
this.CalDataV3.Item4.measAttenPortModule[30],
//    //
this.CalDataV3.Item4.measAttenPortModule[31],
//    //
this.CalDataV3.Item4.measAttenPortModule[32],
//    //
this.CalDataV3.Item4.measAttenPortModule[33],
//    //
this.CalDataV3.Item4.measAttenPortModule[34],
//    //
this.CalDataV3.Item4.measAttenPortModule[35],
//    //
this.CalDataV3.Item4.measAttenPortModule[36],
//    //
this.CalDataV3.Item4.measAttenPortModule[37],
//    //
this.CalDataV3.Item4.measAttenPortModule[38],
//    //
this.CalDataV3.Item4.measAttenPortModule[39],
//    //
this.CalDataV3.Item4.measAttenPortModule[40],
//    //
this.CalDataV3.Item4.measAttenPortModule[41],
//    //
this.CalDataV3.Item4.measAttenPortModule[42],
//    //
this.CalDataV3.Item4.measAttenPortModule[43],
//    //
this.CalDataV3.Item4.measAttenPortModule[44],
//    //
this.CalDataV3.Item4.measAttenPortModule[45]);

//    //                    sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //                    sw.WriteLine("Source,Source Port,Source Port Alias,Measure Port,Measure
Port Alias,Frequency,Level,Filter,Meas Atten,Modulation Type,Modulation File,Duty
Cycle,Comment,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //                    foreach (CalDataVer3 tl in this.CalDataV3.Item1)
//    //                    {
//    //                        sw.Write("{0},", tl.srcSelect);
```

```
//    //                              sw.Write("{0},", tl.srcPort);
//    //                              sw.Write("{0},", tl.srcPortAlias);
//    //                              sw.Write("{0},", tl.measPort);
//    //                              sw.Write("{0},", tl.measPortAlias);
//    //                              sw.Write("{0},", tl.srcFreq);
//    //                              sw.Write("{0},", tl.srcLevel);
//    //                              sw.Write("{0},", tl.measFilter.ToString().ToUpper());
//    //                              sw.Write("{0},", tl.measAtten);
//    //                              sw.Write("{0},", tl.modulationType);
//    //                              sw.Write("{0},", tl.modulationFile);
//    //                              sw.Write("{0},", tl.dutyCycle);
//    //                              sw.Write("{0},", tl.comment);
//    //                              sw.Write(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //                              sw.WriteLine();
//    //                          }
//    //                      }
//    //                      break;
//    //                  }
//    //              case "v4":
//    //                  {


//    //                      using (StreamWriter sw = new StreamWriter(FileName))
//    //                      {
//    //                          // Info
//    //                          sw.WriteLine("Product:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,", this.CalDataV4.Item1.Product);
//    //                          sw.WriteLine("Revision:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,",
this.CalDataV4.Item1.Revision);
//    //                          sw.WriteLine("Test Program:,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");
//    //                          sw.WriteLine("Calibration Version:,v4,,,,,,,,,,,,,,,,,,,,,,,,,,,,");
//    //                          sw.WriteLine("Comment:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,",
this.CalDataV4.Item1.Comment);
//    //                          sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //                          // Cal Settings
//    //                          sw.WriteLine("Calibrate RF Source/Measure:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV4.Item2.SourceMeasurePath));
//    //                          sw.WriteLine("Calibrate Internal Direct Path:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV4.Item2.InternalMatrixPath));
//    //                          sw.WriteLine("Calibrate Noise Source:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV4.Item2.NoiseSource));
```

```
//    //                    sw.WriteLine("Calibrate VNA:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV4.Item2.VNA));

//    //                    sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");
//    //                    sw.WriteLine("Noise Bandwidth (Hz):,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,,,",
this.CalDataV4.Item2.NoiseBandwidth);
//    //                    sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //                    // External Attenuators
//    //
sw.WriteLine(",P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15,P16,SG1,,,,,,,,,,,,");
//    //                    sw.WriteLine("MT-RF100 Attenuation
(dB):,{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},{12},{13},{14},{15},{16},,,,,,,,,,,,",
//    //
this.CalDataV4.Item3.srcAttenPortModule[0],
//    //
this.CalDataV4.Item3.srcAttenPortModule[1],
//    //
this.CalDataV4.Item3.srcAttenPortModule[2],
//    //
this.CalDataV4.Item3.srcAttenPortModule[3],
//    //
this.CalDataV4.Item3.srcAttenPortModule[4],
//    //
this.CalDataV4.Item3.srcAttenPortModule[5],
//    //
this.CalDataV4.Item3.srcAttenPortModule[6],
//    //
this.CalDataV4.Item3.srcAttenPortModule[7],
//    //
this.CalDataV4.Item3.srcAttenPortModule[8],
//    //
this.CalDataV4.Item3.srcAttenPortModule[9],
//    //
this.CalDataV4.Item3.srcAttenPortModule[10],
//    //
this.CalDataV4.Item3.srcAttenPortModule[11],
//    //
this.CalDataV4.Item3.srcAttenPortModule[12],
//    //
this.CalDataV4.Item3.srcAttenPortModule[13],
//    //
```

```
this.CalDataV4.Item3.srcAttenPortModule[14],
//    //
this.CalDataV4.Item3.srcAttenPortModule[15],
//    //
this.CalDataV4.Item3.srcAttenPortModule[16]);

//    //                    sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //
sw.WriteLine(",M1,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11,M12,M13,M14,M15,M16,M17,M18,M1
9,M20,M21,M22,M23,M24,M25,M26,MA,MB,MC,MD");
//    //                    sw.WriteLine("MT-RF200 Attenuation
(dB):,{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},{12},{13},{14},{15},{16},{17},{18},{19},{20},{21},{2
2},{23},{24},{25},{26},{27},{28},{29}",
//    //
this.CalDataV4.Item3.measAttenPortModule[0],
//    //
this.CalDataV4.Item3.measAttenPortModule[1],
//    //
this.CalDataV4.Item3.measAttenPortModule[2],
//    //
this.CalDataV4.Item3.measAttenPortModule[3],
//    //
this.CalDataV4.Item3.measAttenPortModule[4],
//    //
this.CalDataV4.Item3.measAttenPortModule[5],
//    //
this.CalDataV4.Item3.measAttenPortModule[6],
//    //
this.CalDataV4.Item3.measAttenPortModule[7],
//    //
this.CalDataV4.Item3.measAttenPortModule[8],
//    //
this.CalDataV4.Item3.measAttenPortModule[9],
//    //
this.CalDataV4.Item3.measAttenPortModule[10],
//    //
this.CalDataV4.Item3.measAttenPortModule[11],
//    //
this.CalDataV4.Item3.measAttenPortModule[12],
//    //
this.CalDataV4.Item3.measAttenPortModule[13],
```

```
//    //
this.CalDataV4.Item3.measAttenPortModule[14],
//    //
this.CalDataV4.Item3.measAttenPortModule[15],
//    //
this.CalDataV4.Item3.measAttenPortModule[16],
//    //
this.CalDataV4.Item3.measAttenPortModule[17],
//    //
this.CalDataV4.Item3.measAttenPortModule[18],
//    //
this.CalDataV4.Item3.measAttenPortModule[19],
//    //
this.CalDataV4.Item3.measAttenPortModule[20],
//    //
this.CalDataV4.Item3.measAttenPortModule[21],
//    //
this.CalDataV4.Item3.measAttenPortModule[22],
//    //
this.CalDataV4.Item3.measAttenPortModule[23],
//    //
this.CalDataV4.Item3.measAttenPortModule[24],
//    //
this.CalDataV4.Item3.measAttenPortModule[25],
//    //
this.CalDataV4.Item3.measAttenPortModule[26],
//    //
this.CalDataV4.Item3.measAttenPortModule[27],
//    //
this.CalDataV4.Item3.measAttenPortModule[28],
//    //
this.CalDataV4.Item3.measAttenPortModule[29]);

//    //                    sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //                    sw.WriteLine("Source,Source Port,Source Port Alias,Measure Port,Measure
Port Alias,Frequency,Level,Filter,Meas Atten,Modulation Type,Modulation File,Duty
Cycle,Comment,,,,,,,,,,,,,,,,");

//    //                    foreach (CalDataPortModule cd in this.CalDataV4.Item4)
//    //                    {
//    //                        // Corner Case - The VNA Names use a Hypon in their names that can't be
```

```
used in an enum. Therefore I used _ instead. To
//    //                            // be written out correctly we need to convert back to the correct txt.
//    //                            if (cd.srcSelect == SigGen.VNA_1P)
//    //                            {
//    //                                sw.Write("VNA-1P,");
//    //                            }
//    //                            else if (cd.srcSelect == SigGen.VNA_2P)
//    //                            {
//    //                                sw.Write("VNA-2P,");
//    //                            }
//    //                            else
//    //                            {
//    //                                sw.Write("{0},", cd.srcSelect);
//    //                            }
//    //                            sw.Write("{0},", cd.srcPort);
//    //                            sw.Write("{0},", cd.srcPortAlias);
//    //                            sw.Write("{0},", cd.measPort);
//    //                            sw.Write("{0},", cd.measPortAlias);
//    //                            sw.Write("{0},", cd.srcFreq);
//    //                            sw.Write("{0},", cd.srcLevel);
//    //                            sw.Write("{0},", cd.measFilter.ToString().ToUpper());
//    //                            sw.Write("{0},", cd.measAtten);
//    //                            sw.Write("{0},", cd.modulationType);
//    //                            sw.Write("{0},", cd.modulationFile);
//    //                            sw.Write("{0},", cd.dutyCycle);
//    //                            sw.Write("{0},", cd.comment);
//    //                            sw.WriteLine(",,,,,,,,,,,,,,,,");

//    //                        }
//    //                    }
//    //                    break;
//    //                }
//    //            case "v5":
//    //                {


//    //                    using (StreamWriter sw = new StreamWriter(FileName))
//    //                    {
//    //                        // Info
//    //                        sw.WriteLine("Product:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,", this.CalDataV5.Item1.Product);
//    //                        sw.WriteLine("Revision:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,,",
this.CalDataV5.Item1.Revision);
```

```
//	//			sw.WriteLine("Test Program:,,,,,,,,,,,,,,,,,,,,,,,,,,,,");
//	//			sw.WriteLine("Calibration Version:,v5,,,,,,,,,,,,,,,,,,,,,,,,,,,");
//	//			sw.WriteLine("Comment:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,",
this.CalDataV5.Item1.Comment);
//	//			sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//	//			// Cal Settings
//	//			sw.WriteLine("Calibrate RF Source/Measure:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV5.Item2.SourceMeasurePath));
//	//			sw.WriteLine("Calibrate Internal Direct Path:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV5.Item2.InternalMatrixPath));
//	//			sw.WriteLine("Calibrate Noise Source:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV5.Item2.NoiseSource));
//	//			sw.WriteLine("Calibrate VNA:,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,",
ConvertBoolToYN(this.CalDataV5.Item2.VNA));

//	//			sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,");
//	//			sw.WriteLine("Noise Bandwidth (Hz):,{0},,,,,,,,,,,,,,,,,,,,,,,,,,,",
this.CalDataV5.Item2.NoiseBandwidth);
//	//			sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//	//			// External Attenuators
//	//
sw.WriteLine(",P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15,P16,SG1,,,,,,,,,,,");
//	//			sw.WriteLine("MT-RF100 Attenuation
(dB):,{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},{12},{13},{14},{15},{16},,,,,,,,,,,",
//	//
this.CalDataV5.Item3.srcAttenPortModule[0],
//	//
this.CalDataV5.Item3.srcAttenPortModule[1],
//	//
this.CalDataV5.Item3.srcAttenPortModule[2],
//	//
this.CalDataV5.Item3.srcAttenPortModule[3],
//	//
this.CalDataV5.Item3.srcAttenPortModule[4],
//	//
this.CalDataV5.Item3.srcAttenPortModule[5],
//	//
this.CalDataV5.Item3.srcAttenPortModule[6],
//	//
this.CalDataV5.Item3.srcAttenPortModule[7],
```

```
//    //
this.CalDataV5.Item3.srcAttenPortModule[8],
//    //
this.CalDataV5.Item3.srcAttenPortModule[9],
//    //
this.CalDataV5.Item3.srcAttenPortModule[10],
//    //
this.CalDataV5.Item3.srcAttenPortModule[11],
//    //
this.CalDataV5.Item3.srcAttenPortModule[12],
//    //
this.CalDataV5.Item3.srcAttenPortModule[13],
//    //
this.CalDataV5.Item3.srcAttenPortModule[14],
//    //
this.CalDataV5.Item3.srcAttenPortModule[15],
//    //
this.CalDataV5.Item3.srcAttenPortModule[16]);

//    //                    sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //
sw.WriteLine(",M1,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11,M12,M13,M14,M15,M16,M17,M18,M19,M20,M21,M22,M23,M24,M25,M26,MA,MB,MC,MD");
//    //                    sw.WriteLine("MT-RF200 Attenuation (dB):,{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},{12},{13},{14},{15},{16},{17},{18},{19},{20},{21},{22},{23},{24},{25},{26},{27},{28},{29}",
//    //
this.CalDataV5.Item3.measAttenPortModule[0],
//    //
this.CalDataV5.Item3.measAttenPortModule[1],
//    //
this.CalDataV5.Item3.measAttenPortModule[2],
//    //
this.CalDataV5.Item3.measAttenPortModule[3],
//    //
this.CalDataV5.Item3.measAttenPortModule[4],
//    //
this.CalDataV5.Item3.measAttenPortModule[5],
//    //
this.CalDataV5.Item3.measAttenPortModule[6],
//    //
```

```
this.CalDataV5.Item3.measAttenPortModule[7],
//    //
this.CalDataV5.Item3.measAttenPortModule[8],
//    //
this.CalDataV5.Item3.measAttenPortModule[9],
//    //
this.CalDataV5.Item3.measAttenPortModule[10],
//    //
this.CalDataV5.Item3.measAttenPortModule[11],
//    //
this.CalDataV5.Item3.measAttenPortModule[12],
//    //
this.CalDataV5.Item3.measAttenPortModule[13],
//    //
this.CalDataV5.Item3.measAttenPortModule[14],
//    //
this.CalDataV5.Item3.measAttenPortModule[15],
//    //
this.CalDataV5.Item3.measAttenPortModule[16],
//    //
this.CalDataV5.Item3.measAttenPortModule[17],
//    //
this.CalDataV5.Item3.measAttenPortModule[18],
//    //
this.CalDataV5.Item3.measAttenPortModule[19],
//    //
this.CalDataV5.Item3.measAttenPortModule[20],
//    //
this.CalDataV5.Item3.measAttenPortModule[21],
//    //
this.CalDataV5.Item3.measAttenPortModule[22],
//    //
this.CalDataV5.Item3.measAttenPortModule[23],
//    //
this.CalDataV5.Item3.measAttenPortModule[24],
//    //
this.CalDataV5.Item3.measAttenPortModule[25],
//    //
this.CalDataV5.Item3.measAttenPortModule[26],
//    //
this.CalDataV5.Item3.measAttenPortModule[27],
//    //
```

```
this.CalDataV5.Item3.measAttenPortModule[28],
//    //
this.CalDataV5.Item3.measAttenPortModule[29]);

//    //                    sw.WriteLine(",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,");

//    //                    sw.WriteLine("Source,Source Port,Source Port Alias,Measure Port,Measure
Port Alias,Frequency,Level,Filter,Meas Atten,Modulation Type,Modulation File,Marker,Analysis
Slot,Downconverter IF Freq,Downconverter RF Atten,Downconverter IF Atten,Downconverter
Preamp,External Type,Comment,,,,,,,,,,");

//    //                    foreach (CalDataPortModule cd in this.CalDataV5.Item4)
//    //                    {
//    //                        // Corner Case - The VNA Names use a Hypon in their names that can't be
used in an enum. Therefore I used _ instead. To
//    //                        // be written out correctly we need to convert back to the correct txt.
//    //                        if (cd.srcSelect == SigGen.VNA_1P)
//    //                        {
//    //                            sw.Write("VNA-1P,");
//    //                        }
//    //                        else if (cd.srcSelect == SigGen.VNA_2P)
//    //                        {
//    //                            sw.Write("VNA-2P,");
//    //                        }
//    //                        else
//    //                        {
//    //                            sw.Write("{0},", cd.srcSelect);
//    //                        }
//    //                        sw.Write("{0},", cd.srcPort);
//    //                        sw.Write("{0},", cd.srcPortAlias);
//    //                        sw.Write("{0},", cd.measPort);
//    //                        sw.Write("{0},", cd.measPortAlias);
//    //                        sw.Write("{0},", cd.srcFreq);
//    //                        sw.Write("{0},", cd.srcLevel);
//    //                        sw.Write("{0},", cd.measFilter.ToString().ToUpper());
//    //                        sw.Write("{0},", cd.measAtten);
//    //                        sw.Write("{0},", cd.modulationType);
//    //                        sw.Write("{0},", cd.modulationFile);
//    //                        sw.Write("{0},", cd.WaveConfig.Marker);
//    //                        sw.Write("{0},", cd.WaveConfig.AnalysisSlot);
//    //                        sw.Write("{0},", "NaN");
//    //                        sw.Write("{0},", "NaN");
```

```
//    //                        sw.Write("{0},", "NaN");
//    //                        sw.Write("{0},", "NaN");
//    //                        sw.Write("{0},", "NaN");
//    //                        sw.Write("{0},", cd.comment);
//    //                        sw.WriteLine(",,,,,,,,,,,,,,,,");

//    //                    }
//    //                }
//    //                break;
//    //            }
//    //        }
//    //    }
//    //    catch (Exception ex)
//    //    {
//    //        string message = string.Format("An error occured while saving a calibration definition
file {0}. {1}", FileName, ex.Message);
//    //        TestStudioModel.trace.TraceInformation(message);
//    //        throw new CalibrationDefinitionException(message);
//    //    }
//    //}

//    /// <summary>
//    /// Create the OnPropertyChanged method to raise the event.
//    /// </summary>
//    /// <param name="name">Name of the property that's just been updated.</param>
//    public void OnPropertyChanged(string name)
//    {
//        PropertyChangedEventHandler handler = PropertyChanged;

//        if (handler != null)
//        {
//            handler(this, new PropertyChangedEventArgs(name));
//        }
//    }

//    #endregion Public Methods
//}
}
```

CalResultsModel.cs

```
//using MT.Core.Common;
using MT.TestStudio.Exceptions;
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTest.Tools.TestStudio.Model
{
public class CalResultsNoise
{
public string Source { get; set; }
public string SourcePort { get; set; }
public string MeasurePort { get; set; }
public string Frequency { get; set; }
public double ENR { get; set; }
public double Loss { get; set; }
public double NoiseOn { get; set; }
public double NoiseOff { get; set; }
}

public class CalResultsNoiseV4
{
public string Source { get; set; }
public string SourcePort { get; set; }
public string SourcePortAlias { get; set; }
public string MeasurePort { get; set; }
public string MeasurePortAlias { get; set; }
public string Frequency { get; set; }
public double BypassENR { get; set; }
public double BypassNoiseOn { get; set; }
public double BypassNoiseOff { get; set; }
public double A1ENR { get; set; }
public double A1NoiseOn { get; set; }
public double A1NoiseOff { get; set; }
public double A2ENR { get; set; }
public double A2NoiseOn { get; set; }
public double A2NoiseOff { get; set; }
public double A1_A2ENR { get; set; }
public double A1_A2NoiseOn { get; set; }
public double A1_A2NoiseOff { get; set; }
```

```
}

public class CalResultRFV4
{
public string Source { get; set; }
public string SourcePort { get; set; }
public string SourcePortAlias { get; set; }
public string MeasurePort { get; set; }
public string MeasurePortAlias { get; set; }
public double Frequency { get; set; }
public double Level { get; set; }
public string Filter { get; set; }
public double MeasAtten { get; set; }
public string ModulationType { get; set; }
public string ModulationFile { get; set; }
public double DutyCycle { get; set; }
public string Comment { get; set; }
public double SrcCalFactorBypass { get; set; }
public double SrcCalFactorA1 { get; set; }
public double SrcCalFactorA2 { get; set; }
public double SrcCalFactorA3 { get; set; }
public double SrcCalFactorA1_A2 { get; set; }
public double SrcCalFactorA1_A3 { get; set; }
public double SrcCalFactorA2_A3 { get; set; }
public double SrcCalFactorA1_A2_A3 { get; set; }
public double MeasCalFactorBypass { get; set; }
public double MeasCalFactorA1 { get; set; }
public double MeasCalFactorA2 { get; set; }
public double MeasCalFactorA1_A2 { get; set; }
public double MeasCalFactorA4Meas { get; set; }
public double MeasCalFactorA4Meas_A1 { get; set; }
public double MeasCalFactorA4Meas_A2 { get; set; }
public double MeasCalFactorA4Meas_A1_A2 { get; set; }
public double DirCalFactorBypassdivBypass { get; set; }
public double DirCalFactorBypassdivA1 { get; set; }
public double DirCalFactorBypassdivA2 { get; set; }
public double DirCalFactorBypassdivA1_A2 { get; set; }
public double DirCalFactorA1divBypass { get; set; }
public double DirCalFactorA1divA1 { get; set; }
public double DirCalFactorA1divA2 { get; set; }
public double DirCalFactorA1divA1_A2 { get; set; }
public double DirCalFactorA2divBypass { get; set; }
```

```
public double DirCalFactorA2divA1 { get; set; }
public double DirCalFactorA2divA2 { get; set; }
public double DirCalFactorA2divA1_A2 { get; set; }
public double DirCalFactorA3divBypass { get; set; }
public double DirCalFactorA3divA1 { get; set; }
public double DirCalFactorA3divA2 { get; set; }
public double DirCalFactorA3divA1_A2 { get; set; }
public double DirCalFactorA1_A2divBypass { get; set; }
public double DirCalFactorA1_A2divA1 { get; set; }
public double DirCalFactorA1_A2divA2 { get; set; }
public double DirCalFactorA1_A2divA1_A2 { get; set; }
public double DirCalFactorA1_A3divBypass { get; set; }
public double DirCalFactorA1_A3divA1 { get; set; }
public double DirCalFactorA1_A3divA2 { get; set; }
public double DirCalFactorA1_A3divA1_A2 { get; set; }
public double DirCalFactorA2_A3divBypass { get; set; }
public double DirCalFactorA2_A3divA1 { get; set; }
public double DirCalFactorA2_A3divA2 { get; set; }
public double DirCalFactorA2_A3divA1_A2 { get; set; }
public double DirCalFactorA1_A2_A3divBypass { get; set; }
public double DirCalFactorA1_A2_A3divA1 { get; set; }
public double DirCalFactorA1_A2_A3divA2 { get; set; }
public double DirCalFactorA1_A2_A3divA1_A2 { get; set; }
}

public class CalResultRFV5
{
public string Source { get; set; }
public string SourcePort { get; set; }
public string SourcePortAlias { get; set; }
public string MeasurePort { get; set; }
public string MeasurePortAlias { get; set; }
public double Frequency { get; set; }
public double Level { get; set; }
public string Filter { get; set; }
public double MeasAtten { get; set; }
public string ModulationType { get; set; }
public string ModulationFile { get; set; }
public double Marker { get; set; }
public double AnalysisSlot { get; set; }
public double Bandwidth { get; set; }
public double DutyCycle { get; set; }
```

```
public double DownConverterIfFreq { get; set; }
public double DownConverterRfAtten { get; set; }
public double DownConverterIfAtten { get; set; }
public bool   DownConverterPreAmp { get; set; }
public string ExternalType { get; set; }
public string Comment { get; set; }
public double SrcCalFactorBypass { get; set; }
public double SrcCalFactorA1 { get; set; }
public double SrcCalFactorA2 { get; set; }
public double SrcCalFactorA3 { get; set; }
public double SrcCalFactorA1_A2 { get; set; }
public double SrcCalFactorA1_A3 { get; set; }
public double SrcCalFactorA2_A3 { get; set; }
public double SrcCalFactorA1_A2_A3 { get; set; }
public double MeasCalFactorBypass { get; set; }
public double MeasCalFactorA1 { get; set; }
public double MeasCalFactorA2 { get; set; }
public double MeasCalFactorA1_A2 { get; set; }
public double MeasCalFactorA4Meas { get; set; }
public double MeasCalFactorA4Meas_A1 { get; set; }
public double MeasCalFactorA4Meas_A2 { get; set; }
public double MeasCalFactorA4Meas_A1_A2 { get; set; }
public double DirCalFactorBypassdivBypass { get; set; }
public double DirCalFactorBypassdivA1 { get; set; }
public double DirCalFactorBypassdivA2 { get; set; }
public double DirCalFactorBypassdivA1_A2 { get; set; }
public double DirCalFactorA1divBypass { get; set; }
public double DirCalFactorA1divA1 { get; set; }
public double DirCalFactorA1divA2 { get; set; }
public double DirCalFactorA1divA1_A2 { get; set; }
public double DirCalFactorA2divBypass { get; set; }
public double DirCalFactorA2divA1 { get; set; }
public double DirCalFactorA2divA2 { get; set; }
public double DirCalFactorA2divA1_A2 { get; set; }
public double DirCalFactorA3divBypass { get; set; }
public double DirCalFactorA3divA1 { get; set; }
public double DirCalFactorA3divA2 { get; set; }
public double DirCalFactorA3divA1_A2 { get; set; }
public double DirCalFactorA1_A2divBypass { get; set; }
public double DirCalFactorA1_A2divA1 { get; set; }
public double DirCalFactorA1_A2divA2 { get; set; }
public double DirCalFactorA1_A2divA1_A2 { get; set; }
```

```
public double DirCalFactorA1_A3divBypass { get; set; }
public double DirCalFactorA1_A3divA1 { get; set; }
public double DirCalFactorA1_A3divA2 { get; set; }
public double DirCalFactorA1_A3divA1_A2 { get; set; }
public double DirCalFactorA2_A3divBypass { get; set; }
public double DirCalFactorA2_A3divA1 { get; set; }
public double DirCalFactorA2_A3divA2 { get; set; }
public double DirCalFactorA2_A3divA1_A2 { get; set; }
public double DirCalFactorA1_A2_A3divBypass { get; set; }
public double DirCalFactorA1_A2_A3divA1 { get; set; }
public double DirCalFactorA1_A2_A3divA2 { get; set; }
public double DirCalFactorA1_A2_A3divA1_A2 { get; set; }
public double DblrSigGenPwr { get; set; }
public double DblrOutPwr { get; set; }
}

public class CalResultRFV3
{
public string Source { get; set; }
public string SourcePort { get; set; }
public string SourcePortAlias { get; set; }
public string MeasurePort { get; set; }
public string MeasurePortAlias { get; set; }
public double Frequency { get; set; }
public double Level { get; set; }
public string Filter { get; set; }
public double MeasAtten { get; set; }
public string ModulationType { get; set; }
public string ModulationFile { get; set; }
public double DutyCycle { get; set; }
public string Comment { get; set; }
public double SrcCalFactorBypass { get; set; }
public double SrcCalFactor1 { get; set; }
public double SrcCalFactor2 { get; set; }
public double SrcCalFactor3 { get; set; }
public double SrcCalFactor4 { get; set; }
public double SrcCalFactor5 { get; set; }
public double SrcCalFactor6 { get; set; }
public double SrcCalFactor7 { get; set; }
public double SrcCalFactor8 { get; set; }
public double MeasCalFactor1 { get; set; }
public double MeasCalFactor2 { get; set; }
```

```
public double MeasCalFactor3 { get; set; }
public double MeasCalFactor4 { get; set; }
public double DirCalFactor1 { get; set; }
public double DirCalFactor2 { get; set; }
public double DirCalFactor3 { get; set; }
public double DirCalFactor4 { get; set; }

}

public enum CalResultsType
{
RfV3,
RfV4,
RfV5,
NoiseV4
}

//public class CalResultsModel
//{
//    #region Private Member Variables



//    #endregion Private Member Variables

//    #region Constructor

//    /// <summary>
//    /// Constructor for the Cal Results Model.
//    /// </summary>
//    public CalResultsModel()
//    {

//    }

//    #endregion Constructor

//    #region Public Properties

//    /// <summary>
//    /// String to indicate which current CAL file has been loaded.
//    /// </summary>
```

```
//    public string CalVersion { get; set; }

//    public string Product { get; private set; }
//    public string Revision { get; private set; }
//    public string TestProgram { get; private set; }
//    public string Date { get; private set; }
//    public string TesterID { get; private set; }
//    public string Comment { get; private set; }
//    public double NoiseBW { get; private set; }

//   public CalResultsType CalibrationFileType { get; set; }

//    public List<CalResultsNoise> CalibrationResultsNoise { get; set; } = new
List<CalResultsNoise>();
//    public List<CalResultsNoiseV4> CalibrationResultsNoiseV4 { get; set; } = new
List<CalResultsNoiseV4>();
//    public List<CalResultRFV4> CalibrationResultsRFV4 { get; set; } = new
List<CalResultRFV4>();
//    public List<CalResultRFV5> CalibrationResultsRFV5 { get; set; } = new
List<CalResultRFV5>();
//    public List<CalResultRFV3> CalibrationResultsRFV3 { get; set; } = new
List<CalResultRFV3>();
//    public List<double> SourceAttenuatorList { get; set; } = new List<double>();
//    public List<double> MeasureAttenuatorList { get; set; } = new List<double>();
//    public List<double> AmplifierBiasCurrentList { get; set; } = new List<double>();
//    public List<double> AmplifierTemperatureList { get; set; } = new List<double>();

//    #endregion Public Properties

//    #region Public Events / Delegates

//    /// <summary>
//    /// Event of type PropertyChangedEventHandler which is used to implement
INotifyPropertyChanged
//    /// and provide notification of when a property has changed.
//    /// </summary>
//    public event PropertyChangedEventHandler PropertyChanged;

//    #endregion Public Events / Delegates

//    #region Public Methods
```

```
//    /// <summary>
//    /// Method that causes a file to be read in.
//    /// </summary>
//    /// <param name="FileName">The file to be read in.</param>
//    public void LoadCsvDataFile(string FileName)
//    {
//        try
//        {
//            int numberOfRowRequiredToGetHeaderInfo = 20;
//            int startOfData = int.MinValue;
//            bool isRfFile = false;

//            // Read header of file and split by ','
//            var headerInfo = (from lines in
File.ReadLines(FileName).Take(numberOfRowRequiredToGetHeaderInfo)
//                              let columns = lines.Split(',')
//                              select columns).ToList();

//            // Meta Data Stored in Header
//            for (int index = 0; index < headerInfo.Count; index++)
//            {
//                if (headerInfo[index].Length > 1)
//                {
//                    // Product Name
//                    if (headerInfo[index][0].ToUpper() == "PRODUCT:")
//                    {
//                        this.Product = headerInfo[index][1];
//                    }

//                    // Revision
//                    if (headerInfo[index][0].ToUpper() == "REVISION:")
//                    {
//                        this.Revision = headerInfo[index][1];
//                    }

//                    // Test Program
//                    if (headerInfo[index][0].ToUpper() == "TEST PROGRAM:")
//                    {
//                        this.TestProgram = headerInfo[index][1];
//                    }

//                    // Date
```

```
//                if (headerInfo[index][0].ToUpper() == "DATE:")
//                {
//                    this.Date = headerInfo[index][1];
//                }

//                // Tester ID
//                if (headerInfo[index][0].ToUpper() == "TESTER ID:")
//                {
//                    this.TesterID = headerInfo[index][1];
//                }

//                // Comment
//                if (headerInfo[index][0].ToUpper() == "COMMENT:")
//                {
//                    this.Comment = headerInfo[index][1];
//                }

//                // Cal Version
//                if (headerInfo[index][0].ToUpper() == "CALIBRATION VERSION:")
//                {
//                    this.CalVersion = headerInfo[index][1];
//                }

//                // Noise BW
//                if (headerInfo[index][0].ToUpper() == "NOISE BANDWIDTH (HZ):")
//                {
//                    this.NoiseBW = HelperMethods.ConvertStringToDouble(headerInfo[index][1]);
//                }

//                // If we find the "Amplifier BIAS Gain" field then we can conclude it's a RF Cal Data
file.
//                if (headerInfo[index][0].ToUpper() == "AMPLIFIER BIAS CURRENT:")
//                {
//                    isRfFile = true;
//                }

//                // If we find the "Amplifier BIAS Gain" field then we can conclude it's a RF Cal Data
file.
//                if (headerInfo[index][0].ToUpper() == "MT-RF100 ATTENUATION (DB):")
//                {
//                    for (int i = 1; i < headerInfo[index].Length; i++) // Start at one to avoid meta Data.
//                    {
```

```
//
this.SourceAttenuatorList.Add(HelperMethods.ConvertStringToDouble(headerInfo[index][i]));
//                  }
//              }

//              // If we find the "Amplifier BIAS Gain" field then we can conclude it's a RF Cal Data
file.
//              if (headerInfo[index][0].ToUpper() == "MT-RF200 ATTENUATION (DB):")
//              {
//                  for (int i = 1; i < headerInfo[index].Length; i++) // Start at one to avoid meta Data.
//                  {
//
this.MeasureAttenuatorList.Add(HelperMethods.ConvertStringToDouble(headerInfo[index][i]));
//                  }
//              }

//              // If we find the "Amplifier BIAS Gain" field then we can conclude it's a RF Cal Data
file.
//              if (headerInfo[index][0].ToUpper() == "AMPLIFIER BIAS CURRENT:")
//              {
//                  for (int i = 1; i < headerInfo[index].Length; i++) // Start at one to avoid meta Data.
//                  {
//
this.AmplifierBiasCurrentList.Add(HelperMethods.ConvertStringToDouble(headerInfo[index][i]));
//                  }
//              }

//              // If we find the "Amplifier BIAS Gain" field then we can conclude it's a RF Cal Data
file.
//              if (headerInfo[index][0].ToUpper() == "AMPLIFIER BIAS TEMPERATURE:")
//              {
//                  for (int i = 1; i < headerInfo[index].Length; i++) // Start at one to avoid meta Data.
//                  {
//
this.AmplifierTemperatureList.Add(HelperMethods.ConvertStringToDouble(headerInfo[index][i]));
//                  }
//              }



//              // Start Of Data
//              if (headerInfo[index][0].ToUpper() == "SOURCE")
//              {
```

```
//                  startOfData = index;
//              }
//          }
//      }

//      // Determine the Cal Results File Type.
//      if ( isRfFile == true )
//      {
//          if ( CalVersion == "v4")
//          {
//              this.CalibrationFileType = CalResultsType.RfV4;
//          }
//          else if (CalVersion == "v3")
//          {
//              this.CalibrationFileType = CalResultsType.RfV3;
//          }
//          else if (CalVersion == "v5")
//          {
//              this.CalibrationFileType = CalResultsType.RfV5;
//          }
//          else
//          {
//              throw new CalibrationResultsException("Unsupported calibration results file.");
//          }
//      }
//      else
//      {
//          if (CalVersion == "v4")
//          {
//              this.CalibrationFileType = CalResultsType.NoiseV4;
//          }
//          else
//          {
//              throw new CalibrationResultsException("Unsupported calibration results file.");
//          }
//      }

//      // Now read in the data.
//      if (startOfData > 0)
//      {
//          var data = (from lines in File.ReadLines(FileName).Skip(startOfData)
//                      let columns = lines.Split(',')
```

```
//                      select columns).ToList();


//              switch (this.CalibrationFileType)
//              {
//                  case CalResultsType.NoiseV4:
//                      {
//                          for (int index = 1; index < data.Count; index++) // Start from 1 to skip over
header
//                          {
//                              if (data[index].Length == 18)
//                              {

//                                  CalResultsNoiseV4 tmp = new CalResultsNoiseV4();

//                                  tmp.Source = data[index][0].ToString();
//                                  tmp.SourcePort = data[index][1].ToString();
//                                  tmp.SourcePortAlias = data[index][2].ToString();
//                                  tmp.MeasurePort = data[index][3].ToString();
//                                  tmp.MeasurePortAlias = data[index][4].ToString();
//                                  tmp.Frequency = data[index][5].ToString();
//                                  tmp.BypassENR =
HelperMethods.ConvertStringToDouble(data[index][6]);
//                                  tmp.BypassNoiseOff =
HelperMethods.ConvertStringToDouble(data[index][7]);
//                                  tmp.BypassNoiseOn =
HelperMethods.ConvertStringToDouble(data[index][8]);
//                                  tmp.A1ENR = HelperMethods.ConvertStringToDouble(data[index][9]);
//                                  tmp.A1NoiseOff =
HelperMethods.ConvertStringToDouble(data[index][10]);
//                                  tmp.A1NoiseOn =
HelperMethods.ConvertStringToDouble(data[index][11]);
//                                  tmp.A2ENR = HelperMethods.ConvertStringToDouble(data[index][12]);
//                                  tmp.A2NoiseOff =
HelperMethods.ConvertStringToDouble(data[index][13]);
//                                  tmp.A2NoiseOn =
HelperMethods.ConvertStringToDouble(data[index][14]);
//                                  tmp.A1_A2ENR =
HelperMethods.ConvertStringToDouble(data[index][15]);
//                                  tmp.A1_A2NoiseOff =
HelperMethods.ConvertStringToDouble(data[index][16]);
//                                  tmp.A1_A2NoiseOn =
```

```
HelperMethods.ConvertStringToDouble(data[index][17]);

//                          // Add to list
//                          CalibrationResultsNoiseV4.Add(tmp);
//                      }
//                      else
//                      {
//                          throw new CalibrationResultsException("V4 Noise Calibration Results File
should contain 18 columns of data but this file had " + data[index].Length + ".");
//                      }

//                  }
//                  break;
//              }
//              case CalResultsType.RfV4:
//                  {
//                  for (int index = 1; index < data.Count; index++) // Start from 1 to skip over
header
//                  {
//                      if (data[index].Length == 61)
//                      {

//                          CalResultRFV4 tmp = new CalResultRFV4();

//                          tmp.Source = data[index][0].ToString();
//                          tmp.SourcePort = data[index][1].ToString();
//                          tmp.SourcePortAlias = data[index][2].ToString();
//                          tmp.MeasurePort = data[index][3].ToString();
//                          tmp.MeasurePortAlias = data[index][4].ToString();
//                          tmp.Frequency = HelperMethods.ConvertStringToDouble(data[index][5]);
//                          tmp.Level = HelperMethods.ConvertStringToDouble(data[index][6]);
//                          tmp.Filter = data[index][7];
//                          tmp.MeasAtten = HelperMethods.ConvertStringToDouble(data[index][8]);
//                          tmp.ModulationType = data[index][9];
//                          tmp.ModulationFile = data[index][10];
//                          tmp.DutyCycle = HelperMethods.ConvertStringToDouble(data[index][11]);
//                          tmp.Comment = data[index][12];


//                          tmp.SrcCalFactorBypass =
HelperMethods.ConvertStringToDouble(data[index][13]);
//                          tmp.SrcCalFactorA1 =
```

```
HelperMethods.ConvertStringToDouble(data[index][14]);
//                      tmp.SrcCalFactorA2 =
HelperMethods.ConvertStringToDouble(data[index][15]);
//                      tmp.SrcCalFactorA3 =
HelperMethods.ConvertStringToDouble(data[index][16]);
//                      tmp.SrcCalFactorA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][17]);
//                      tmp.SrcCalFactorA1_A3 =
HelperMethods.ConvertStringToDouble(data[index][18]);
//                      tmp.SrcCalFactorA2_A3 =
HelperMethods.ConvertStringToDouble(data[index][19]);
//                      tmp.SrcCalFactorA1_A2_A3 =
HelperMethods.ConvertStringToDouble(data[index][20]);

//                      tmp.MeasCalFactorBypass =
HelperMethods.ConvertStringToDouble(data[index][21]);
//                      tmp.MeasCalFactorA1 =
HelperMethods.ConvertStringToDouble(data[index][22]);
//                      tmp.MeasCalFactorA2 =
HelperMethods.ConvertStringToDouble(data[index][23]);
//                      tmp.MeasCalFactorA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][24]);
//                      tmp.MeasCalFactorA4Meas =
HelperMethods.ConvertStringToDouble(data[index][25]);
//                      tmp.MeasCalFactorA4Meas_A1 =
HelperMethods.ConvertStringToDouble(data[index][26]);
//                      tmp.MeasCalFactorA4Meas_A2 =
HelperMethods.ConvertStringToDouble(data[index][27]);
//                      tmp.MeasCalFactorA4Meas_A1_A2 =
HelperMethods.ConvertStringToDouble(data[index][28]);

//                      tmp.DirCalFactorBypassdivBypass =
HelperMethods.ConvertStringToDouble(data[index][29]);
//                      tmp.DirCalFactorBypassdivA1 =
HelperMethods.ConvertStringToDouble(data[index][30]);
//                      tmp.DirCalFactorBypassdivA2 =
HelperMethods.ConvertStringToDouble(data[index][31]);
//                      tmp.DirCalFactorBypassdivA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][32]);

//                      tmp.DirCalFactorA1divBypass =
HelperMethods.ConvertStringToDouble(data[index][33]);
```

```
//                      tmp.DirCalFactorA1divA1 =
HelperMethods.ConvertStringToDouble(data[index][34]);
//                      tmp.DirCalFactorA1divA2 =
HelperMethods.ConvertStringToDouble(data[index][35]);
//                      tmp.DirCalFactorA1divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][36]);

//                      tmp.DirCalFactorA2divBypass =
HelperMethods.ConvertStringToDouble(data[index][37]);
//                      tmp.DirCalFactorA2divA1 =
HelperMethods.ConvertStringToDouble(data[index][38]);
//                      tmp.DirCalFactorA2divA2 =
HelperMethods.ConvertStringToDouble(data[index][39]);
//                      tmp.DirCalFactorA2divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][40]);

//                      tmp.DirCalFactorA3divBypass =
HelperMethods.ConvertStringToDouble(data[index][41]);
//                      tmp.DirCalFactorA3divA1 =
HelperMethods.ConvertStringToDouble(data[index][42]);
//                      tmp.DirCalFactorA3divA2 =
HelperMethods.ConvertStringToDouble(data[index][43]);
//                      tmp.DirCalFactorA3divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][44]);

//                      tmp.DirCalFactorA1_A2divBypass =
HelperMethods.ConvertStringToDouble(data[index][45]);
//                      tmp.DirCalFactorA1_A2divA1 =
HelperMethods.ConvertStringToDouble(data[index][46]);
//                      tmp.DirCalFactorA1_A2divA2 =
HelperMethods.ConvertStringToDouble(data[index][47]);
//                      tmp.DirCalFactorA1_A2divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][48]);

//                      tmp.DirCalFactorA1_A3divBypass =
HelperMethods.ConvertStringToDouble(data[index][49]);
//                      tmp.DirCalFactorA1_A3divA1 =
HelperMethods.ConvertStringToDouble(data[index][50]);
//                      tmp.DirCalFactorA1_A3divA2 =
HelperMethods.ConvertStringToDouble(data[index][51]);
//                      tmp.DirCalFactorA1_A3divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][52]);
```

```
//                          tmp.DirCalFactorA2_A3divBypass =
HelperMethods.ConvertStringToDouble(data[index][53]);
//                          tmp.DirCalFactorA2_A3divA1 =
HelperMethods.ConvertStringToDouble(data[index][54]);
//                          tmp.DirCalFactorA2_A3divA2 =
HelperMethods.ConvertStringToDouble(data[index][55]);
//                          tmp.DirCalFactorA2_A3divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][56]);

//                          tmp.DirCalFactorA1_A2_A3divBypass =
HelperMethods.ConvertStringToDouble(data[index][57]);
//                          tmp.DirCalFactorA1_A2_A3divA1 =
HelperMethods.ConvertStringToDouble(data[index][58]);
//                          tmp.DirCalFactorA1_A2_A3divA2 =
HelperMethods.ConvertStringToDouble(data[index][59]);
//                          tmp.DirCalFactorA1_A2_A3divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][60]);



//                          // Add to list
//                          CalibrationResultsRFV4.Add(tmp);
//                      }
//                      else
//                      {
//                          throw new CalibrationResultsException("V4 Noise Calibration Results File
should contain 18 columns of data but this file had " + data[index].Length + ".");
//                      }

//                  }
//                  break;
//              }
//              case CalResultsType.RfV3:
//                  {
//                      for (int index = 1; index < data.Count; index++) // Start from 1 to skip over
header
//                      {
//                          if (data[index].Length == 30)
//                          {

//                              CalResultRFV3 tmp = new CalResultRFV3();
```

```
//                      tmp.Source = data[index][0].ToString();
//                      tmp.SourcePort = data[index][1].ToString();
//                      tmp.SourcePortAlias = data[index][2].ToString();
//                      tmp.MeasurePort = data[index][3].ToString();
//                      tmp.MeasurePortAlias = data[index][4].ToString();
//                      tmp.Frequency = HelperMethods.ConvertStringToDouble(data[index][5]);
//                      tmp.Level = HelperMethods.ConvertStringToDouble(data[index][6]);
//                      tmp.Filter = data[index][7];
//                      tmp.MeasAtten = HelperMethods.ConvertStringToDouble(data[index][8]);
//                      tmp.ModulationType = data[index][9];
//                      tmp.ModulationFile = data[index][10];
//                      tmp.DutyCycle = HelperMethods.ConvertStringToDouble(data[index][11]);
//                      tmp.Comment = data[index][12];

//                      tmp.SrcCalFactor1 =
HelperMethods.ConvertStringToDouble(data[index][13]);
//                      tmp.SrcCalFactor2 =
HelperMethods.ConvertStringToDouble(data[index][14]);
//                      tmp.SrcCalFactor3 =
HelperMethods.ConvertStringToDouble(data[index][15]);
//                      tmp.SrcCalFactor4 =
HelperMethods.ConvertStringToDouble(data[index][16]);
//                      tmp.SrcCalFactor5 =
HelperMethods.ConvertStringToDouble(data[index][17]);
//                      tmp.SrcCalFactor6 =
HelperMethods.ConvertStringToDouble(data[index][18]);
//                      tmp.SrcCalFactor7 =
HelperMethods.ConvertStringToDouble(data[index][19]);
//                      tmp.SrcCalFactor8 =
HelperMethods.ConvertStringToDouble(data[index][20]);

//                      tmp.MeasCalFactor1 =
HelperMethods.ConvertStringToDouble(data[index][21]);
//                      tmp.MeasCalFactor2 =
HelperMethods.ConvertStringToDouble(data[index][22]);
//                      tmp.MeasCalFactor3 =
HelperMethods.ConvertStringToDouble(data[index][23]);
//                      tmp.MeasCalFactor4 =
HelperMethods.ConvertStringToDouble(data[index][24]);

//                      tmp.DirCalFactor1 =
HelperMethods.ConvertStringToDouble(data[index][25]);
```

```
//                          tmp.DirCalFactor2 =
HelperMethods.ConvertStringToDouble(data[index][26]);
//                          tmp.DirCalFactor3 =
HelperMethods.ConvertStringToDouble(data[index][27]);
//                          tmp.DirCalFactor4 =
HelperMethods.ConvertStringToDouble(data[index][28]);

//                          // Add to list
//                          CalibrationResultsRFV3.Add(tmp);
//                      }
//                      else
//                      {
//                          throw new CalibrationResultsException("V3 RF Calibration Results File
should contain 30 columns of data but this file had " + data[index].Length + ".");
//                      }

//                  }
//                  break;
//              }
//              case CalResultsType.RfV5:
//                  {
//                  for (int index = 1; index < data.Count; index++) // Start from 1 to skip over
header
//                  {
//                      if (data[index].Length == 71)
//                      {

//                          CalResultRFV5 tmp = new CalResultRFV5();

//                          tmp.Source = data[index][0].ToString();
//                          tmp.SourcePort = data[index][1].ToString();
//                          tmp.SourcePortAlias = data[index][2].ToString();
//                          tmp.MeasurePort = data[index][3].ToString();
//                          tmp.MeasurePortAlias = data[index][4].ToString();
//                          tmp.Frequency = HelperMethods.ConvertStringToDouble(data[index][5]);
//                          tmp.Level = HelperMethods.ConvertStringToDouble(data[index][6]);
//                          tmp.Filter = data[index][7];
//                          tmp.MeasAtten = HelperMethods.ConvertStringToDouble(data[index][8]);
//                          tmp.ModulationType = data[index][9];
//                          tmp.ModulationFile = data[index][10];
//                          tmp.Marker = HelperMethods.ConvertStringToDouble(data[index][11]);
//                          tmp.AnalysisSlot =
```

```
HelperMethods.ConvertStringToDouble(data[index][12]);
//                      tmp.Bandwidth = HelperMethods.ConvertStringToDouble(data[index][13]);
//                      tmp.DutyCycle = HelperMethods.ConvertStringToDouble(data[index][14]);
//                      tmp.DownConverterIfFreq =
HelperMethods.ConvertStringToDouble(data[index][15]);
//                      tmp.DownConverterRfAtten =
HelperMethods.ConvertStringToDouble(data[index][16]);
//                      tmp.DownConverterIfAtten =
HelperMethods.ConvertStringToDouble(data[index][17]);
//                      tmp.DownConverterPreAmp =
HelperMethods.ConvertStringToBool(data[index][18]);
//                      tmp.ExternalType = data[index][19];
//                      tmp.Comment = data[index][20];

//                      tmp.SrcCalFactorBypass =
HelperMethods.ConvertStringToDouble(data[index][21]);
//                      tmp.SrcCalFactorA1 =
HelperMethods.ConvertStringToDouble(data[index][22]);
//                      tmp.SrcCalFactorA2 =
HelperMethods.ConvertStringToDouble(data[index][23]);
//                      tmp.SrcCalFactorA3 =
HelperMethods.ConvertStringToDouble(data[index][24]);
//                      tmp.SrcCalFactorA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][25]);
//                      tmp.SrcCalFactorA1_A3 =
HelperMethods.ConvertStringToDouble(data[index][26]);
//                      tmp.SrcCalFactorA2_A3 =
HelperMethods.ConvertStringToDouble(data[index][27]);
//                      tmp.SrcCalFactorA1_A2_A3 =
HelperMethods.ConvertStringToDouble(data[index][28]);

//                      tmp.MeasCalFactorBypass =
HelperMethods.ConvertStringToDouble(data[index][29]);
//                      tmp.MeasCalFactorA1 =
HelperMethods.ConvertStringToDouble(data[index][30]);
//                      tmp.MeasCalFactorA2 =
HelperMethods.ConvertStringToDouble(data[index][31]);
//                      tmp.MeasCalFactorA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][32]);
//                      tmp.MeasCalFactorA4Meas =
HelperMethods.ConvertStringToDouble(data[index][33]);
//                      tmp.MeasCalFactorA4Meas_A1 =
```

```
HelperMethods.ConvertStringToDouble(data[index][34]);
//                      tmp.MeasCalFactorA4Meas_A2 =
HelperMethods.ConvertStringToDouble(data[index][35]);
//                      tmp.MeasCalFactorA4Meas_A1_A2 =
HelperMethods.ConvertStringToDouble(data[index][36]);

//                      tmp.DirCalFactorBypassdivBypass =
HelperMethods.ConvertStringToDouble(data[index][37]);
//                      tmp.DirCalFactorBypassdivA1 =
HelperMethods.ConvertStringToDouble(data[index][38]);
//                      tmp.DirCalFactorBypassdivA2 =
HelperMethods.ConvertStringToDouble(data[index][39]);
//                      tmp.DirCalFactorBypassdivA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][40]);

//                      tmp.DirCalFactorA1divBypass =
HelperMethods.ConvertStringToDouble(data[index][41]);
//                      tmp.DirCalFactorA1divA1 =
HelperMethods.ConvertStringToDouble(data[index][42]);
//                      tmp.DirCalFactorA1divA2 =
HelperMethods.ConvertStringToDouble(data[index][43]);
//                      tmp.DirCalFactorA1divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][44]);

//                      tmp.DirCalFactorA2divBypass =
HelperMethods.ConvertStringToDouble(data[index][45]);
//                      tmp.DirCalFactorA2divA1 =
HelperMethods.ConvertStringToDouble(data[index][46]);
//                      tmp.DirCalFactorA2divA2 =
HelperMethods.ConvertStringToDouble(data[index][47]);
//                      tmp.DirCalFactorA2divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][48]);

//                      tmp.DirCalFactorA3divBypass =
HelperMethods.ConvertStringToDouble(data[index][49]);
//                      tmp.DirCalFactorA3divA1 =
HelperMethods.ConvertStringToDouble(data[index][50]);
//                      tmp.DirCalFactorA3divA2 =
HelperMethods.ConvertStringToDouble(data[index][51]);
//                      tmp.DirCalFactorA3divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][52]);
```

```
//                      tmp.DirCalFactorA1_A2divBypass =
HelperMethods.ConvertStringToDouble(data[index][53]);
//                      tmp.DirCalFactorA1_A2divA1 =
HelperMethods.ConvertStringToDouble(data[index][54]);
//                      tmp.DirCalFactorA1_A2divA2 =
HelperMethods.ConvertStringToDouble(data[index][55]);
//                      tmp.DirCalFactorA1_A2divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][56]);

//                      tmp.DirCalFactorA1_A3divBypass =
HelperMethods.ConvertStringToDouble(data[index][57]);
//                      tmp.DirCalFactorA1_A3divA1 =
HelperMethods.ConvertStringToDouble(data[index][58]);
//                      tmp.DirCalFactorA1_A3divA2 =
HelperMethods.ConvertStringToDouble(data[index][59]);
//                      tmp.DirCalFactorA1_A3divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][60]);

//                      tmp.DirCalFactorA2_A3divBypass =
HelperMethods.ConvertStringToDouble(data[index][61]);
//                      tmp.DirCalFactorA2_A3divA1 =
HelperMethods.ConvertStringToDouble(data[index][62]);
//                      tmp.DirCalFactorA2_A3divA2 =
HelperMethods.ConvertStringToDouble(data[index][63]);
//                      tmp.DirCalFactorA2_A3divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][64]);

//                      tmp.DirCalFactorA1_A2_A3divBypass =
HelperMethods.ConvertStringToDouble(data[index][65]);
//                      tmp.DirCalFactorA1_A2_A3divA1 =
HelperMethods.ConvertStringToDouble(data[index][66]);
//                      tmp.DirCalFactorA1_A2_A3divA2 =
HelperMethods.ConvertStringToDouble(data[index][67]);
//                      tmp.DirCalFactorA1_A2_A3divA1_A2 =
HelperMethods.ConvertStringToDouble(data[index][68]);

//                      tmp.DblrSigGenPwr =
HelperMethods.ConvertStringToDouble(data[index][69]);
//                      tmp.DblrOutPwr =
HelperMethods.ConvertStringToDouble(data[index][70]);

//                      // Add to list
```

```
//                          CalibrationResultsRFV5.Add(tmp);
//                      }
//                      else
//                      {
//                          throw new CalibrationResultsException("V5 Noise Calibration Results File
should contain 71 columns of data but this file had " + data[index].Length + ".");
//                      }

//                  }
//                  break;
//              }
//          default:
//              {
//                  for (int index = 1; index < data.Count; index++) // Start from 1 to skip over
header
//                  {
//                      if (data[index].Length > 1)
//                      {

//                          CalResultsNoise tmp = new CalResultsNoise();

//                          tmp.Source = data[index][0].ToString();
//                          tmp.SourcePort = data[index][1].ToString();
//                          tmp.MeasurePort = data[index][2].ToString();
//                          tmp.Frequency = data[index][3].ToString();
//                          tmp.ENR = HelperMethods.ConvertStringToDouble(data[index][4]);
//                          tmp.Loss = HelperMethods.ConvertStringToDouble(data[index][5]);
//                          tmp.NoiseOn = HelperMethods.ConvertStringToDouble(data[index][6]);
//                          tmp.NoiseOff = HelperMethods.ConvertStringToDouble(data[index][7]);

//                          // Add to list
//                          CalibrationResultsNoise.Add(tmp);

//                      }
//                  }
//                  break;
//              }
//      }



//      // Notify the View-Model that the product config data has been read in.
```

```
//                OnPropertyChanged("CsvDataLoaded");
//            }
//            else
//            {
//                throw new CalibrationResultsException("Start of data not found.");
//            }
//        }
//        catch (Exception ex)
//        {
//            string message = string.Format("An error occured while loading a test condition file {0}.
{1}", FileName, ex.Message);
//            TestStudioModel.trace.TraceInformation(message);
//            throw new CalibrationResultsException(message);
//        }

//    }

//    /// <summary>
//    /// Create the OnPropertyChanged method to raise the event.
//    /// </summary>
//    /// <param name="name">Name of the property that's just been updated.</param>
//    public void OnPropertyChanged(string name)
//    {
//        PropertyChangedEventHandler handler = PropertyChanged;

//        if (handler != null)
//        {
//            handler(this, new PropertyChangedEventArgs(name));
//        }
//    }

//    #endregion Public Methods
//}
}
```

Cal_Import.cs

```
using System;
using System.IO;
using System.Linq;
using System.Xml.Linq;
using System.Windows.Forms;
using System.Collections.Generic;
```

```
using System.Text.RegularExpressions;
using MT.TestStudio.Enums;
//using MT.Core.Common;
using MT.TestStudio.Exceptions;

//namespace MT.TesterDriver
namespace MT.TestStudio.Calibration
{
//=================================================================================
// The following classes / methods should be used for all future projects to simplify
// data collection and import for Port Module Calibration, User Calibration, and test
// applications.
//=================================================================================

public class Info
{
public string TesterId { get; set; }
public string Product { get; set; }
public string Revision { get; set; }
public string Program { get; set; }
public string CalVersion { get; set; }
public string Date { get; set; }
public string Comment { get; set; }
public string CalStatus { get; set; }
}

public class AmpData
{
public const double gain1 = 19.5; // Typical gain (dB) for HMC8410
public const double gain2 = 13.0; // Typical gain (dB) for HMC637ALP5E
public const double gain3 = 15.5; // Typical gain (dB) for HMC637ALP5E replacement

public double[] srcPortAmpBias = new double[5];
public double[] srcPortAmpTemp = new double[5];

public double[] measPortAmpBias = new double[2];
public double[] measPortAmpTemp = new double[2];
}

public class PortModuleAttenuation
```

```
{
public double[] srcAtten = new double[17];
public double[] measAtten = new double[30];
}

public class CalSettings
{
public bool PowerMeterAvailable { get; set; }
public bool SourceMeasurePath { get; set; }
public bool MeasurePathLNA { get; set; }
public bool InternalMatrixPath { get; set; }
public bool NoiseSource { get; set; }
public bool DCSense { get; set; }
public bool VNA { get; set; }
public double NoiseBandwidth { get; set; }
}

public class CalDataPortModule
{
public int record = 0;
public bool copy = false;
public SigGen srcSelect;
public SrcPort srcPort;
public MeasPort measPort;
public MeasFilt measFilter;
public string comment = "";
public string srcPortAlias = "";
public string measPortAlias = "";
public string modulationType = "";
public string modulationFile = "";
public double srcFreq = 0;
public double srcLevel = -100;
public double dutyCycle = 100;
public double measAtten = 0;
public double[] srcCalFactor = new double[8];
public double[] measCalFactor = new double[8];
public double[] calCalFactor = new double[32];
public double noiseCalFactorSrc = 0;
public double noiseCalFactorSrcPath = 0;
public double noiseCalFactorOn = 0;
public double noiseCalFactorOff = 0;
public double noiseCalFactorLoss = 0;
```

```
public double noiseCalFactorENR = 0;

public double[] portModuleNoiseCalFactorOn = new double[4];
public double[] portModuleNoiseCalFactorOff = new double[4];
public double[] portModuleNoiseCalFactorENR = new double[4];

public bool usesSigGen1 = false;
public bool usesSigGen2 = false;
public bool usesNoiseSource = false;

//// Waveform marker information
//MarkerData iqsMarkerData;

// Added for dynamic waveform calibration (User Cal v5.0 and up)
public WaveformSettings WaveConfig;

// Adde for 5G adjacent chanel power calibration (User Cal v5.0 and up)
public double[] srcCalFactorUpper = new double[8];
public double[] measCalFactorUpper = new double[8];
public double[] calCalFactorUpper = new double[32];
public double[] srcCalFactorLower = new double[8];
public double[] measCalFactorLower = new double[8];
public double[] calCalFactorLower = new double[32];
}


//======================== Port Module Calibration (System)
========================

public class SourceModuleSystemCal
{
bool writeToConsole = true;

public Info info = new Info();
public AmpData ampData = new AmpData();
public Dictionary<Tuple<double, SrcAmpConfig>, double> ampGainData = new
Dictionary<Tuple<double, SrcAmpConfig>, double>();
public Dictionary<Tuple<double, SrcPort, SrcAmpConfig>, double> sourceCalData = new
Dictionary<Tuple<double, SrcPort, SrcAmpConfig>, double>();
public Dictionary<Tuple<double, MeasPort, SrcMeasAmpConfig>, double> measureCalData =
new Dictionary<Tuple<double, MeasPort, SrcMeasAmpConfig>, double>();
public Dictionary<Tuple<double, MeasPort, SrcMeasAmpConfig>, double> couplerInCalData =
```

```
new Dictionary<Tuple<double, MeasPort, SrcMeasAmpConfig>, double>();
public Dictionary<Tuple<double, MeasPort, SrcMeasAmpConfig>, double> couplerOutCalData =
new Dictionary<Tuple<double, MeasPort, SrcMeasAmpConfig>, double>();
public List<double> freqList = new List<double>();

public int ReadCalDataFile()
{
const string calDataFile = @"C:\MerlinTest\System\Calibration\Port
Module\APS100_RF_Source_Port_Module_Cal_Data_Baseline.csv";

if (writeToConsole) Console.WriteLine("Loading source port module calibration data file: {0}\n",
calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "WARNING: Port module calibration data file not found:" + calDataFile + "\n";
if (writeToConsole) Console.WriteLine(message);
//MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1; // Cal data file not found
}

// Check if file is locked
bool fileLocked = true;
do
{
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
fileLocked = false;
}
catch (IOException ex)
{
MessageBox.Show(ex.Message + "\n\nPlease close file to continue...", "Merlin Test
Technologies", MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1);
}
} while (fileLocked == true);
```

```
// Read cal data from file
ImportCalData(calDataFile);

// Convert dictionary values to array
int counter = 0;
var calDataArray = sourceCalData.Keys.ToArray();
do
{
freqList.Add(calDataArray[counter].Item1);
counter += 17;
} while (counter < calDataArray.Length);

if (this.info.CalStatus.ToUpper() == "FAIL") return 0; // Cal data file found (with cal failures)
else return 1; // Cal data file found (no cal failures)
}

public int ReadCalDataFile(string calDataFile)
{
if (writeToConsole) Console.WriteLine("Loading source port module calibration data file: {0}\n",
calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "WARNING: Port module calibration data file not found:" + calDataFile + "\n";
if (writeToConsole) Console.WriteLine(message);
//MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1; // Cal data file not found
}

// Check if file is locked
bool fileLocked = true;
do
{
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
fileLocked = false;
}
```

```
catch (IOException ex)
{
MessageBox.Show(ex.Message + "\n\nPlease close file to continue...", "Merlin Test
Technologies", MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1);
}
} while (fileLocked == true);

// Read cal data from file
ImportCalData(calDataFile);

// Convert dictionary values to array
int counter = 0;
var calDataArray = sourceCalData.Keys.ToArray();
do
{
freqList.Add(calDataArray[counter].Item1);
counter += 17;
} while (counter < calDataArray.Length);

if (this.info.CalStatus.ToUpper() == "FAIL") return 0; // Cal data file found (with cal failures)
else return 1; // Cal data file found (no cal failures)
}

private int ImportCalData(string calDataFile)
{
string line = "";
double freq = 0;
double calData = 0;

SrcPort srcPort = SrcPort.NONE;
SrcAmpConfig srcAmpConfig = SrcAmpConfig.None;

MeasPort measPort = MeasPort.NONE;
SrcMeasAmpConfig measAmpConfig = SrcMeasAmpConfig.Bypass;

Tuple<double, SrcAmpConfig> AmpGainKey = new Tuple<double, SrcAmpConfig>(freq,
srcAmpConfig);
Tuple<double, SrcPort, SrcAmpConfig> SourcePortKey = new Tuple<double, SrcPort,
SrcAmpConfig>(freq, srcPort, srcAmpConfig);
Tuple<double, MeasPort, SrcMeasAmpConfig> MeasurePortKey = new Tuple<double, MeasPort,
SrcMeasAmpConfig>(freq, measPort, measAmpConfig);
```

```
// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "ERROR: Calibration configuration file not found!\n" + calDataFile;
if (writeToConsole) Console.WriteLine(message);
//MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Check if file is locked
bool fileLocked = true;
do
{
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
fileLocked = false;
}
catch (IOException ex)
{
MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
}
} while (fileLocked == true);

// Read cal data file contents
try
{
using (StreamReader stream = new StreamReader(calDataFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "TESTER ID:")
{
info.TesterId = line.Split(',')[1];
}
```

```
else if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "HARDWARE REVISION:")
{
info.Revision = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "DATE:")
{
info.Date = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "COMMENT:")
{
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "CALIBRATION RESULT:")
{
info.CalStatus = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "AMPLIFIER BIAS CURRENT:")
{
ampData.srcPortAmpBias[0] = double.Parse(line.Split(',')[1]);
ampData.srcPortAmpBias[1] = double.Parse(line.Split(',')[2]);
ampData.srcPortAmpBias[2] = double.Parse(line.Split(',')[3]);
ampData.srcPortAmpBias[3] = double.Parse(line.Split(',')[4]);
ampData.srcPortAmpBias[4] = double.Parse(line.Split(',')[5]);
}
else if (tmpString.ToUpper() == "AMPLIFIER BIAS TEMPERATURE:")
{
ampData.srcPortAmpTemp[0] = double.Parse(line.Split(',')[1]);
ampData.srcPortAmpTemp[1] = double.Parse(line.Split(',')[2]);
ampData.srcPortAmpTemp[2] = double.Parse(line.Split(',')[3]);
ampData.srcPortAmpTemp[3] = double.Parse(line.Split(',')[4]);
ampData.srcPortAmpTemp[4] = double.Parse(line.Split(',')[5]);
}
else if (tmpString.ToUpper() == "GAIN")
{
```

```
line = stream.ReadLine();

while ((line = stream.ReadLine()) != "")
{
freq = double.Parse(line.Split(',')[0]);

// A1, A2, A3
for (int ampConfigIndex = 0; ampConfigIndex < 7; ampConfigIndex++)
{
if (ampConfigIndex == 0) srcAmpConfig = SrcAmpConfig.A1;
else if (ampConfigIndex == 1) srcAmpConfig = SrcAmpConfig.A2;
else if (ampConfigIndex == 2) srcAmpConfig = SrcAmpConfig.A3;
else if (ampConfigIndex == 3) srcAmpConfig = SrcAmpConfig.A1_A2;
else if (ampConfigIndex == 4) srcAmpConfig = SrcAmpConfig.A1_A3;
else if (ampConfigIndex == 5) srcAmpConfig = SrcAmpConfig.A2_A3;
else if (ampConfigIndex == 6) srcAmpConfig = SrcAmpConfig.A1_A2_A3;

calData = double.Parse(line.Split(',')[ampConfigIndex + 1]);
AmpGainKey = Tuple.Create(freq, srcAmpConfig);
ampGainData.Add(AmpGainKey, calData);
}

// A4_MEAS
calData = double.Parse(line.Split(',')[8]);
AmpGainKey = Tuple.Create(freq, SrcAmpConfig.A4_MEAS);
ampGainData.Add(AmpGainKey, calData);

// A1_SG2
calData = double.Parse(line.Split(',')[9]);
AmpGainKey = Tuple.Create(freq, SrcAmpConfig.A1_SG2);
ampGainData.Add(AmpGainKey, calData);
}
}
else if (tmpString.ToUpper() == "P1-P16/SG1")
{
line = stream.ReadLine();

while ((line = stream.ReadLine()) != "")
{
freq = double.Parse(line.Split(',')[0]);

for (int srcPortIndex = 0; srcPortIndex < 17; srcPortIndex++)
```

```
{
if (srcPortIndex == 0) srcPort = SrcPort.P1;
else if (srcPortIndex == 1) srcPort = SrcPort.P2;
else if (srcPortIndex == 2) srcPort = SrcPort.P3;
else if (srcPortIndex == 3) srcPort = SrcPort.P4;
else if (srcPortIndex == 4) srcPort = SrcPort.P5;
else if (srcPortIndex == 5) srcPort = SrcPort.P6;
else if (srcPortIndex == 6) srcPort = SrcPort.P7;
else if (srcPortIndex == 7) srcPort = SrcPort.P8;
else if (srcPortIndex == 8) srcPort = SrcPort.P9;
else if (srcPortIndex == 9) srcPort = SrcPort.P10;
else if (srcPortIndex == 10) srcPort = SrcPort.P11;
else if (srcPortIndex == 11) srcPort = SrcPort.P12;
else if (srcPortIndex == 12) srcPort = SrcPort.P13;
else if (srcPortIndex == 13) srcPort = SrcPort.P14;
else if (srcPortIndex == 14) srcPort = SrcPort.P15;
else if (srcPortIndex == 15) srcPort = SrcPort.P16;
else if (srcPortIndex == 16) srcPort = SrcPort.SG1_OUT;

calData = double.Parse(line.Split(',')[srcPortIndex + 1]);
SourcePortKey = Tuple.Create(freq, srcPort, SrcAmpConfig.Bypass);
sourceCalData.Add(SourcePortKey, calData);
}
}
}
else if (tmpString.ToUpper() == "P1-P16/MA-MD")
{
line = stream.ReadLine();

while ((line = stream.ReadLine()) != "")
{
freq = double.Parse(line.Split(',')[0]);

for (int measPortIndex = 0; measPortIndex < 20; measPortIndex++)
{
if (measPortIndex == 0) measPort = MeasPort.P1;
else if (measPortIndex == 1) measPort = MeasPort.P2;
else if (measPortIndex == 2) measPort = MeasPort.P3;
else if (measPortIndex == 3) measPort = MeasPort.P4;
else if (measPortIndex == 4) measPort = MeasPort.P5;
else if (measPortIndex == 5) measPort = MeasPort.P6;
else if (measPortIndex == 6) measPort = MeasPort.P7;
```

```
else if (measPortIndex == 7) measPort = MeasPort.P8;
else if (measPortIndex == 8) measPort = MeasPort.P9;
else if (measPortIndex == 9) measPort = MeasPort.P10;
else if (measPortIndex == 10) measPort = MeasPort.P11;
else if (measPortIndex == 11) measPort = MeasPort.P12;
else if (measPortIndex == 12) measPort = MeasPort.P13;
else if (measPortIndex == 13) measPort = MeasPort.P14;
else if (measPortIndex == 14) measPort = MeasPort.P15;
else if (measPortIndex == 15) measPort = MeasPort.P16;
else if (measPortIndex == 16) measPort = MeasPort.MA;
else if (measPortIndex == 17) measPort = MeasPort.MB;
else if (measPortIndex == 18) measPort = MeasPort.MC;
else if (measPortIndex == 19) measPort = MeasPort.MD;

calData = double.Parse(line.Split(',')[measPortIndex + 1]);
MeasurePortKey = Tuple.Create(freq, measPort, SrcMeasAmpConfig.Bypass);
measureCalData.Add(MeasurePortKey, calData);
}
}
}
else if (tmpString.ToUpper() == "COUPLER_IN")
{
line = stream.ReadLine();

while ((line = stream.ReadLine()) != "")
{
freq = double.Parse(line.Split(',')[0]);

// P1_Fwd/P1_Rev
calData = double.Parse(line.Split(',')[1]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P1_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[2]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P1_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[3]);

// P2_Fwd/P2_Rev
calData = double.Parse(line.Split(',')[4]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P2_Fwd, SrcMeasAmpConfig.A4_MEAS);
```

```
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[5]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P2_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[6]);

// P3_Fwd/P3_Rev
calData = double.Parse(line.Split(',')[7]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P3_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[8]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P3_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[9]);

// P4_Fwd/P4_Rev
calData = double.Parse(line.Split(',')[10]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P4_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[11]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P4_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[12]);

// P5_Fwd/P5_Rev
calData = double.Parse(line.Split(',')[13]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P5_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[14]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P5_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[15]);

// P6_Fwd/P6_Rev
```

```
calData = double.Parse(line.Split(',')[16]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P6_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[17]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P6_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[18]);

// P7_Fwd/P7_Rev
calData = double.Parse(line.Split(',')[19]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P7_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[20]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P7_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[21]);

// P8_Fwd/P8_Rev
calData = double.Parse(line.Split(',')[22]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P8_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[23]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P8_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerInCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[24]);
}
}
else if (tmpString.ToUpper() == "COUPLER_OUT")
{
line = stream.ReadLine();

while ((line = stream.ReadLine()) != null) // EOF
{
freq = double.Parse(line.Split(',')[0]);

// P1_Fwd/P1_Rev
```

```
calData = double.Parse(line.Split(',')[1]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P1_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[2]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P1_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[3]);

// P2_Fwd/P2_Rev
calData = double.Parse(line.Split(',')[4]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P2_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[5]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P2_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[6]);

// P3_Fwd/P3_Rev
calData = double.Parse(line.Split(',')[7]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P3_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[8]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P3_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[9]);

// P4_Fwd/P4_Rev
calData = double.Parse(line.Split(',')[10]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P4_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[11]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P4_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[12]);
```

```
// P5_Fwd/P5_Rev
calData = double.Parse(line.Split(',')[13]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P5_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[14]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P5_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[15]);

// P6_Fwd/P6_Rev
calData = double.Parse(line.Split(',')[16]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P6_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[17]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P6_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[18]);

// P7_Fwd/P7_Rev
calData = double.Parse(line.Split(',')[19]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P7_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[20]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P7_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[21]);

// P8_Fwd/P8_Rev
calData = double.Parse(line.Split(',')[22]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P8_Fwd, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);

calData = double.Parse(line.Split(',')[23]);
MeasurePortKey = Tuple.Create(freq, MeasPort.P8_Rev, SrcMeasAmpConfig.A4_MEAS);
couplerOutCalData.Add(MeasurePortKey, calData);
```

```
calData = double.Parse(line.Split(',')[24]);
}
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
return -1;
}

return 0;
}

public bool CalFrequencyExists(double freq, SrcAmpConfig amp)
{
bool isAvailable = true;

try
{
double calValue = ampGainData[new Tuple<double, SrcAmpConfig>(freq, amp)];
}
catch
{
isAvailable = false;
}

return isAvailable;
}

public bool CalFrequencyExists(double freq, SrcPort port, SrcAmpConfig amp)
{
bool isAvailable = true;

try
{
double calValue = sourceCalData[new Tuple<double, SrcPort, SrcAmpConfig>(freq, port, amp)];
}
catch
{
```

```
isAvailable = false;
}

return isAvailable;
}

public bool CalFrequencyExists(double freq, MeasPort port, SrcMeasAmpConfig amp)
{
bool isAvailable = true;

try
{
if (port == MeasPort.P1 || port == MeasPort.P2 || port == MeasPort.P3 || port == MeasPort.P4 ||
port == MeasPort.P5 || port == MeasPort.P6 || port == MeasPort.P7 || port == MeasPort.P8 || port
== MeasPort.P9 || port == MeasPort.P10 || port == MeasPort.P11 || port == MeasPort.P12 || port
== MeasPort.P13 || port == MeasPort.P14 || port == MeasPort.P15 || port == MeasPort.P16 || port
== MeasPort.MA || port == MeasPort.MB || port == MeasPort.MC || port == MeasPort.MD)
{
double calValue = measureCalData[new Tuple<double, MeasPort, SrcMeasAmpConfig>(freq,
port, amp)];
}
else if (port == MeasPort.P1_Fwd || port == MeasPort.P2_Fwd || port == MeasPort.P3_Fwd || port
== MeasPort.P4_Fwd || port == MeasPort.P5_Fwd || port == MeasPort.P6_Fwd || port ==
MeasPort.P7_Fwd || port == MeasPort.P8_Fwd)
{
double calValue = couplerOutCalData[new Tuple<double, MeasPort, SrcMeasAmpConfig>(freq,
port, amp)];
}
else if (port == MeasPort.P1_Rev || port == MeasPort.P2_Rev || port == MeasPort.P3_Rev || port
== MeasPort.P4_Rev || port == MeasPort.P5_Rev || port == MeasPort.P6_Rev || port ==
MeasPort.P7_Rev || port == MeasPort.P8_Rev)
{
double calValue = couplerInCalData[new Tuple<double, MeasPort, SrcMeasAmpConfig>(freq,
port, amp)];
}
}
catch
{
isAvailable = false;
}

return isAvailable;
```

```
}

public double GetInterpolatedAmpGain(SrcAmpConfig ampConfig, double frequency)
{
double lowerFreq = 0;
double upperFreq = 0;
double ampGain = double.NaN;

long freq = (long)Math.Round(frequency);
if (freq < 0 || freq > 6e9) return ampGain;

// Determine if frequency is below the lowest calibration frequency available
if (freq <= freqList[0])
{
ampGain = ampGainData[new Tuple<double, SrcAmpConfig>(freqList[0], ampConfig)];
return ampGain;
}

// Determine if frequency is above the highest calibration frequency available
if (freq >= freqList[freqList.Count - 1])
{
ampGain = ampGainData[new Tuple<double, SrcAmpConfig>(freqList[freqList.Count - 1],
ampConfig)];
return ampGain;
}

// Find the closest calibration frequencies available
for (int i = 0; i < freqList.Count; i++)
{
if (freqList[i] > freq)
{
upperFreq = freqList[i];
lowerFreq = freqList[i - 1];
break;
}
}

// Get interpolated calibration value
double lowerLoss = ampGainData[new Tuple<double, SrcAmpConfig>(lowerFreq, ampConfig)];
double upperLoss = ampGainData[new Tuple<double, SrcAmpConfig>(upperFreq, ampConfig)];
double m = (upperLoss - lowerLoss) / (upperFreq - lowerFreq); // m = delta y / delta x
double b = lowerLoss - (m * lowerFreq); // b = y - mx
```

```
ampGain = m * freq + b; // y = mx + b

return ampGain;
}

public double GetInterpolatedSourcePathLoss(SrcPort port, double frequency)
{
double lowerFreq = 0;
double upperFreq = 0;
double pathLoss = double.NaN;

long freq = (long)Math.Round(frequency);
if (freq < 0 || freq > 6e9) return pathLoss;

// Determine if frequency is below the lowest calibration frequency available
if (freq <= freqList[0])
{
pathLoss = sourceCalData[new Tuple<double, SrcPort, SrcAmpConfig>(freqList[0], port,
SrcAmpConfig.Bypass)];
return pathLoss;
}

// Determine if frequency is above the highest calibration frequency available
if (freq >= freqList[freqList.Count - 1])
{
pathLoss = sourceCalData[new Tuple<double, SrcPort, SrcAmpConfig>(freqList[freqList.Count -
1], port, SrcAmpConfig.Bypass)];
return pathLoss;
}

// Find the closest calibration frequencies available
for (int i = 0; i < freqList.Count; i++)
{
if (freqList[i] > freq)
{
upperFreq = freqList[i];
lowerFreq = freqList[i - 1];
break;
}
}

// Get interpolated calibration value
```

```
double lowerLoss = sourceCalData[new Tuple<double, SrcPort, SrcAmpConfig>(lowerFreq, port,
SrcAmpConfig.Bypass)];
double upperLoss = sourceCalData[new Tuple<double, SrcPort, SrcAmpConfig>(upperFreq, port,
SrcAmpConfig.Bypass)];
double m = (upperLoss - lowerLoss) / (upperFreq - lowerFreq); // m = delta y / delta x
double b = lowerLoss - (m * lowerFreq); // b = y - mx
pathLoss = m * freq + b; // y = mx + b

return pathLoss;
}

public double GetInterpolatedMeasurePathLoss(MeasPort port, double frequency)
{
double lowerFreq = 0;
double upperFreq = 0;
double pathLoss = double.NaN;

long freq = (long)Math.Round(frequency);
if (freq < 0 || freq > 6e9) return pathLoss;

// Determine if frequency is below the lowest calibration frequency available
if (freq <= freqList[0])
{
pathLoss = measureCalData[new Tuple<double, MeasPort, SrcMeasAmpConfig>(freqList[0], port,
SrcMeasAmpConfig.Bypass)];
return pathLoss;
}

// Determine if frequency is above the highest calibration frequency available
if (freq >= freqList[freqList.Count - 1])
{
pathLoss = measureCalData[new Tuple<double, MeasPort,
SrcMeasAmpConfig>(freqList[freqList.Count - 1], port, SrcMeasAmpConfig.Bypass)];
return pathLoss;
}

// Find the closest calibration frequencies available
for (int i = 0; i < freqList.Count; i++)
{
if (freqList[i] > freq)
{
upperFreq = freqList[i];
```

```
lowerFreq = freqList[i - 1];
break;
}
}

// Get interpolated calibration value
double lowerLoss = measureCalData[new Tuple<double, MeasPort,
SrcMeasAmpConfig>(lowerFreq, port, SrcMeasAmpConfig.Bypass)];
double upperLoss = measureCalData[new Tuple<double, MeasPort,
SrcMeasAmpConfig>(upperFreq, port, SrcMeasAmpConfig.Bypass)];
double m = (upperLoss - lowerLoss) / (upperFreq - lowerFreq); // m = delta y / delta x
double b = lowerLoss - (m * lowerFreq); // b = y - mx
pathLoss = m * freq + b; // y = mx + b

return pathLoss;
}

public double GetInterpolatedCouplerInPathLoss(MeasPort port, double frequency)
{
double lowerFreq = 0;
double upperFreq = 0;
double pathLoss = double.NaN;

long freq = (long)Math.Round(frequency);
if (freq < 0 || freq > 6e9) return pathLoss;

// Determine if frequency is below the lowest calibration frequency available
if (freq <= freqList[0])
{
pathLoss = couplerInCalData[new Tuple<double, MeasPort, SrcMeasAmpConfig>(freqList[0], port,
SrcMeasAmpConfig.Bypass)];
return pathLoss;
}

// Determine if frequency is above the highest calibration frequency available
if (freq >= freqList[freqList.Count - 1])
{
pathLoss = couplerInCalData[new Tuple<double, MeasPort,
SrcMeasAmpConfig>(freqList[freqList.Count - 1], port, SrcMeasAmpConfig.Bypass)];
return pathLoss;
}
```

```
// Find the closest calibration frequencies available
for (int i = 0; i < freqList.Count; i++)
{
if (freqList[i] > freq)
{
upperFreq = freqList[i];
lowerFreq = freqList[i - 1];
break;
}
}

// Get interpolated calibration value
double lowerLoss = couplerInCalData[new Tuple<double, MeasPort,
SrcMeasAmpConfig>(lowerFreq, port, SrcMeasAmpConfig.Bypass)];
double upperLoss = couplerInCalData[new Tuple<double, MeasPort,
SrcMeasAmpConfig>(upperFreq, port, SrcMeasAmpConfig.Bypass)];
double m = (upperLoss - lowerLoss) / (upperFreq - lowerFreq); // m = delta y / delta x
double b = lowerLoss - (m * lowerFreq); // b = y - mx
pathLoss = m * freq + b; // y = mx + b

return pathLoss;
}

public double GetInterpolatedCouplerOutPathLoss(MeasPort port, double frequency)
{
double lowerFreq = 0;
double upperFreq = 0;
double pathLoss = double.NaN;

long freq = (long)Math.Round(frequency);
if (freq < 0 || freq > 6e9) return pathLoss;

// Determine if frequency is below the lowest calibration frequency available
if (freq <= freqList[0])
{
pathLoss = couplerOutCalData[new Tuple<double, MeasPort, SrcMeasAmpConfig>(freqList[0],
port, SrcMeasAmpConfig.Bypass)];
return pathLoss;
}

// Determine if frequency is above the highest calibration frequency available
if (freq >= freqList[freqList.Count - 1])
```

```
{
pathLoss = couplerOutCalData[new Tuple<double, MeasPort,
SrcMeasAmpConfig>(freqList[freqList.Count - 1], port, SrcMeasAmpConfig.Bypass)];
return pathLoss;
}

// Find the closest calibration frequencies available
for (int i = 0; i < freqList.Count; i++)
{
if (freqList[i] > freq)
{
upperFreq = freqList[i];
lowerFreq = freqList[i - 1];
break;
}
}

// Get interpolated calibration value
double lowerLoss = couplerOutCalData[new Tuple<double, MeasPort,
SrcMeasAmpConfig>(lowerFreq, port, SrcMeasAmpConfig.Bypass)];
double upperLoss = couplerOutCalData[new Tuple<double, MeasPort,
SrcMeasAmpConfig>(upperFreq, port, SrcMeasAmpConfig.Bypass)];
double m = (upperLoss - lowerLoss) / (upperFreq - lowerFreq); // m = delta y / delta x
double b = lowerLoss - (m * lowerFreq); // b = y - mx
pathLoss = m * freq + b; // y = mx + b

return pathLoss;
}
}

public class MeasureModuleSystemCal
{
bool writeToConsole = true;

public Info info = new Info();
public AmpData ampData = new AmpData();
public Dictionary<Tuple<double, MeasAmpConfig>, double> ampGainData = new
Dictionary<Tuple<double, MeasAmpConfig>, double>();
public Dictionary<Tuple<double, MeasAtten>, double> attenData = new Dictionary<Tuple<double,
MeasAtten>, double>();
public Dictionary<Tuple<double, MeasFilt>, double> filterData = new Dictionary<Tuple<double,
MeasFilt>, double>();
```

```
public Dictionary<Tuple<double, MeasPort, MeasAmpConfig>, double> measureCalData = new
Dictionary<Tuple<double, MeasPort, MeasAmpConfig>, double>();
public List<double> freqList = new List<double>();

public int ReadCalDataFile()
{
const string calDataFile = @"C:\MerlinTest\System\Calibration\Port
Module\APS100_RF_Measure_Port_Module_Cal_Data_Baseline.csv";

if (writeToConsole) Console.WriteLine("Loading measure port module calibration data file: {0}\n",
calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "WARNING: Port module calibration data file not found:" + calDataFile + "\n";
if (writeToConsole) Console.WriteLine(message);
//MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1; // Cal data file not found
}

// Check if file is locked
bool fileLocked = true;
do
{
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
fileLocked = false;
}
catch (IOException ex)
{
MessageBox.Show(ex.Message + "\n\nPlease close file to continue...", "Merlin Test
Technologies", MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1);
}
} while (fileLocked == true);

// Read cal data from file
```

```
ImportCalData(calDataFile);

// Convert dictionary values to array
int counter = 0;
var calDataArray = measureCalData.Keys.ToArray();
do
{
freqList.Add(calDataArray[counter].Item1);
counter += 27;
} while (counter < calDataArray.Length);

if (this.info.CalStatus.ToUpper() == "FAIL") return 0; // Cal data file found (with cal failures)
else return 1; // Cal data file found (no cal failures)
}

public int ReadCalDataFile(string calDataFile)
{
if (writeToConsole) Console.WriteLine("Loading measure port module calibration data file: {0}\n",
calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "WARNING: Port module calibration data file not found:" + calDataFile + "\n";
if (writeToConsole) Console.WriteLine(message);
//MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1; // Cal data file not found
}

// Check if file is locked
bool fileLocked = true;
do
{
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
fileLocked = false;
}
catch (IOException ex)
```

```
{
MessageBox.Show(ex.Message + "\n\nPlease close file to continue...", "Merlin Test
Technologies", MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1);
}
} while (fileLocked == true);

// Read cal data from file
ImportCalData(calDataFile);

// Convert dictionary values to array
int counter = 0;
var calDataArray = measureCalData.Keys.ToArray();
do
{
freqList.Add(calDataArray[counter].Item1);
counter += 27;
} while (counter < calDataArray.Length);

if (this.info.CalStatus.ToUpper() == "FAIL") return 0; // Cal data file found (with cal failures)
else return 1; // Cal data file found (no cal failures)
}

private int ImportCalData(string calDataFile)
{
string line = "";
double freq = 0;
double calData = 0;

MeasPort measPort = MeasPort.NONE;
MeasAmpConfig measAmpConfig = MeasAmpConfig.Bypass;
MeasAtten attenConfig = MeasAtten.Ref;
MeasFilt filterConfig = MeasFilt.NONE;

Tuple<double, MeasAmpConfig> AmpGainKey = new Tuple<double, MeasAmpConfig>(freq,
measAmpConfig);
Tuple<double, MeasAtten> AttenKey = new Tuple<double, MeasAtten>(freq, attenConfig);
Tuple<double, MeasFilt> FilterKey = new Tuple<double, MeasFilt>(freq, filterConfig);
Tuple<double, MeasPort, MeasAmpConfig> MeasurePortKey = new Tuple<double, MeasPort,
MeasAmpConfig>(freq, measPort, measAmpConfig);

// Check if file exists
```

```
if (!File.Exists(calDataFile))
{
string message = "ERROR: Calibration configuration file not found!\n" + calDataFile;
if (writeToConsole) Console.WriteLine(message);
//MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Check if file is locked
bool fileLocked = true;
do
{
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
fileLocked = false;
}
catch (IOException ex)
{
MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
}
} while (fileLocked == true);

// Read cal data file contents
try
{
using (StreamReader stream = new StreamReader(calDataFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "TESTER ID:")
{
info.TesterId = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "PRODUCT:")
{
```

```
info.Product = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "HARDWARE REVISION:")
{
info.Revision = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "DATE:")
{
info.Date = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "COMMENT:")
{
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "CALIBRATION RESULT:")
{
info.CalStatus = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "AMPLIFIER BIAS CURRENT:")
{
ampData.measPortAmpBias[0] = double.Parse(line.Split(',')[1]);
ampData.measPortAmpBias[1] = double.Parse(line.Split(',')[2]);
}
else if (tmpString.ToUpper() == "AMPLIFIER BIAS TEMPERATURE:")
{
ampData.measPortAmpTemp[0] = double.Parse(line.Split(',')[2]);
ampData.measPortAmpTemp[1] = double.Parse(line.Split(',')[2]);
}
else if (tmpString.ToUpper() == "GAIN/ATTENUATION")
{
line = stream.ReadLine();

while ((line = stream.ReadLine()) != "")
{
freq = double.Parse(line.Split(',')[0]);

// A1, A2
for (int ampConfigIndex = 0; ampConfigIndex < 3; ampConfigIndex++)
```

```
{
if (ampConfigIndex == 0) measAmpConfig = MeasAmpConfig.A1;
else if (ampConfigIndex == 1) measAmpConfig = MeasAmpConfig.A2;
else if (ampConfigIndex == 2) measAmpConfig = MeasAmpConfig.A1_A2;

calData = double.Parse(line.Split(',')[ampConfigIndex + 1]);
AmpGainKey = Tuple.Create(freq, measAmpConfig);
ampGainData.Add(AmpGainKey, calData);
}

// Attenuator
for (int attenConfigIndex = 1; attenConfigIndex < 9; attenConfigIndex++)
{
if (attenConfigIndex == 0) attenConfig = MeasAtten.Ref;
else if (attenConfigIndex == 1) attenConfig = MeasAtten.A_p25dB;
else if (attenConfigIndex == 2) attenConfig = MeasAtten.A_p5dB;
else if (attenConfigIndex == 3) attenConfig = MeasAtten.A_1dB;
else if (attenConfigIndex == 4) attenConfig = MeasAtten.A_2dB;
else if (attenConfigIndex == 5) attenConfig = MeasAtten.A_4dB;
else if (attenConfigIndex == 6) attenConfig = MeasAtten.A_8dB;
else if (attenConfigIndex == 7) attenConfig = MeasAtten.A_16dB;
else if (attenConfigIndex == 8) attenConfig = MeasAtten.A_32dB;

calData = double.Parse(line.Split(',')[attenConfigIndex + 3]);
AttenKey = Tuple.Create(freq, attenConfig);
attenData.Add(AttenKey, calData);
}
}
}
else if (tmpString.ToUpper() == "M1-M26/CAL")
{
line = stream.ReadLine();

while ((line = stream.ReadLine()) != "")
{
freq = double.Parse(line.Split(',')[0]);

for (int measPortIndex = 0; measPortIndex < 27; measPortIndex++)
{
if (measPortIndex == 0) measPort = MeasPort.M1;
else if (measPortIndex == 1) measPort = MeasPort.M2;
else if (measPortIndex == 2) measPort = MeasPort.M3;
```

```
else if (measPortIndex == 3) measPort = MeasPort.M4;
else if (measPortIndex == 4) measPort = MeasPort.M5;
else if (measPortIndex == 5) measPort = MeasPort.M6;
else if (measPortIndex == 6) measPort = MeasPort.M7;
else if (measPortIndex == 7) measPort = MeasPort.M8;
else if (measPortIndex == 8) measPort = MeasPort.M9;
else if (measPortIndex == 9) measPort = MeasPort.M10;
else if (measPortIndex == 10) measPort = MeasPort.M11;
else if (measPortIndex == 11) measPort = MeasPort.M12;
else if (measPortIndex == 12) measPort = MeasPort.M13;
else if (measPortIndex == 13) measPort = MeasPort.M14;
else if (measPortIndex == 14) measPort = MeasPort.M15;
else if (measPortIndex == 15) measPort = MeasPort.M16;
else if (measPortIndex == 16) measPort = MeasPort.M17;
else if (measPortIndex == 17) measPort = MeasPort.M18;
else if (measPortIndex == 18) measPort = MeasPort.M19;
else if (measPortIndex == 19) measPort = MeasPort.M20;
else if (measPortIndex == 20) measPort = MeasPort.M21;
else if (measPortIndex == 21) measPort = MeasPort.M22;
else if (measPortIndex == 22) measPort = MeasPort.M23;
else if (measPortIndex == 23) measPort = MeasPort.M24;
else if (measPortIndex == 24) measPort = MeasPort.M25;
else if (measPortIndex == 25) measPort = MeasPort.M26;
else if (measPortIndex == 26) measPort = MeasPort.CAL;

calData = double.Parse(line.Split(',')[measPortIndex + 1]);
MeasurePortKey = Tuple.Create(freq, measPort, MeasAmpConfig.Bypass);
measureCalData.Add(MeasurePortKey, calData);
}
}
}
else if (tmpString.ToUpper() == "FILTER")
{
line = stream.ReadLine();

while ((line = stream.ReadLine()) != "")
{
freq = double.Parse(line.Split(',')[0]);

// FILT1, FILT2, FILT3
for (int filterIndex = 0; filterIndex < 3; filterIndex++)
{
```

```
if (filterIndex == 0) filterConfig = MeasFilt.FILT1;
else if (filterIndex == 1) filterConfig = MeasFilt.FILT2;
else if (filterIndex == 2) filterConfig = MeasFilt.FILT3;

calData = double.Parse(line.Split(',')[filterIndex + 1]);
FilterKey = Tuple.Create(freq, filterConfig);
filterData.Add(FilterKey, calData);
}
}
}
else if (tmpString.ToUpper() == "NOISE")
{
line = stream.ReadLine();

while ((line = stream.ReadLine()) != null) // EOF
{
freq = double.Parse(line.Split(',')[0]);

for (int noiseIndex = 0; noiseIndex < 2; noiseIndex++)
{
// Does not import noise calibration data
}
}
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
return -1;
}

return 0;
}

public bool CalFrequencyExists(double freq, MeasAmpConfig amp)
{
bool isAvailable = true;

try
{
```

```
double calValue = ampGainData[new Tuple<double, MeasAmpConfig>(freq, amp)];
}
catch
{
isAvailable = false;
}

return isAvailable;
}

public bool CalFrequencyExists(double freq, MeasPort port, MeasAmpConfig amp)
{
bool isAvailable = true;

try
{
double calValue = measureCalData[new Tuple<double, MeasPort, MeasAmpConfig>(freq, port,
amp)];
}
catch
{
isAvailable = false;
}

return isAvailable;
}

public double GetInterpolatedAmpGain(MeasAmpConfig ampConfig, double frequency)
{
double lowerFreq = 0;
double upperFreq = 0;
double ampGain = double.NaN;

long freq = (long)Math.Round(frequency);
if (freq < 0 || freq > 6e9) return ampGain;

// Determine if frequency is below the lowest calibration frequency available
if (freq <= freqList[0])
{
ampGain = ampGainData[new Tuple<double, MeasAmpConfig>(freqList[0], ampConfig)];
return ampGain;
}
```

```
// Determine if frequency is above the highest calibration frequency available
if (freq >= freqList[freqList.Count - 1])
{
ampGain = ampGainData[new Tuple<double, MeasAmpConfig>(freqList[freqList.Count - 1],
ampConfig)];
return ampGain;
}

// Find the closest calibration frequencies available
for (int i = 0; i < freqList.Count; i++)
{
if (freqList[i] > freq)
{
upperFreq = freqList[i];
lowerFreq = freqList[i - 1];
break;
}
}

// Get interpolated calibration value
double lowerLoss = ampGainData[new Tuple<double, MeasAmpConfig>(lowerFreq, ampConfig)];
double upperLoss = ampGainData[new Tuple<double, MeasAmpConfig>(upperFreq, ampConfig)];
double m = (upperLoss - lowerLoss) / (upperFreq - lowerFreq); // m = delta y / delta x
double b = lowerLoss - (m * lowerFreq); // b = y - mx
ampGain = m * freq + b; // y = mx + b

return ampGain;
}

public double GetInterpolatedMeasurePathLoss(MeasPort port, double frequency)
{
double lowerFreq = 0;
double upperFreq = 0;
double pathLoss = double.NaN;

long freq = (long)Math.Round(frequency);
if (freq < 0 || freq > 6e9) return pathLoss;

// Determine if frequency is below the lowest calibration frequency available
if (freq <= freqList[0])
{
```

```
pathLoss = measureCalData[new Tuple<double, MeasPort, MeasAmpConfig>(freqList[0], port,
MeasAmpConfig.Bypass)];
return pathLoss;
}

// Determine if frequency is above the highest calibration frequency available
if (freq >= freqList[freqList.Count - 1])
{
pathLoss = measureCalData[new Tuple<double, MeasPort,
MeasAmpConfig>(freqList[freqList.Count - 1], port, MeasAmpConfig.Bypass)];
return pathLoss;
}

// Find the closest calibration frequencies available
for (int i = 0; i < freqList.Count; i++)
{
if (freqList[i] > freq)
{
upperFreq = freqList[i];
lowerFreq = freqList[i - 1];
break;
}
}

// Get interpolated calibration value
double lowerLoss = measureCalData[new Tuple<double, MeasPort, MeasAmpConfig>(lowerFreq,
port, MeasAmpConfig.Bypass)];
double upperLoss = measureCalData[new Tuple<double, MeasPort,
MeasAmpConfig>(upperFreq, port, MeasAmpConfig.Bypass)];
double m = (upperLoss - lowerLoss) / (upperFreq - lowerFreq); // m = delta y / delta x
double b = lowerLoss - (m * lowerFreq); // b = y - mx
pathLoss = m * freq + b; // y = mx + b

return pathLoss;
}
}



//========================== Port Module Calibration (User)
==========================

public class PortModuleUserCalSourceConfig
```

```
{
public double Frequency { get; set; }
public SigGen SigGen { get; set; }
public SrcPort SrcPort { get; set; }
public string Modulation { get; set; }
public double Level { get; set; }   // Optional key
public string Waveform { get; set; } // Optoinal key
}

public class PortModuleUserCalSourceData
{
public double Bypass { get; set; }
public double A1 { get; set; }
public double A2 { get; set; }
public double A3 { get; set; }
public double A1_A2 { get; set; }
public double A1_A3 { get; set; }
public double A2_A3 { get; set; }
public double A1_A2_A3 { get; set; }
}

public class PortModuleUserCalMeasureConfig
{
public double Frequency { get; set; }
public SigGen SigGen { get; set; }
public MeasPort MeasPort { get; set; }
public MeasFilt Attenuator { get; set; }
public string Modulation { get; set; }
public double Level { get; set; }    // Optional key
public string Waveform { get; set; } // Optoinal key
}

public class PortModuleUserCalMeasureData
{
public double Bypass { get; set; }
public double A1 { get; set; }
public double A2 { get; set; }
public double A1_A2 { get; set; }
public double A4_MEAS { get; set; }
public double A4_MEAS_A1 { get; set; }
public double A4_MEAS_A2 { get; set; }
public double A4_MEAS_A1_A2 { get; set; }
```

```
}

public class PortModuleUserCalDirectData
{
public double Bypass { get; set; }
public double Bypass_A1 { get; set; }
public double Bypass_A2 { get; set; }
public double Bypass_A1A2 { get; set; }

public double A1_Bypass { get; set; }
public double A1_A1 { get; set; }
public double A1_A2 { get; set; }
public double A1_A1A2 { get; set; }

public double A2_Bypass { get; set; }
public double A2_A1 { get; set; }
public double A2_A2 { get; set; }
public double A2_A1A2 { get; set; }

public double A3_Bypass { get; set; }
public double A3_A1 { get; set; }
public double A3_A2 { get; set; }
public double A3_A1A2 { get; set; }

public double A1A2_Bypass { get; set; }
public double A1A2_A1 { get; set; }
public double A1A2_A2 { get; set; }
public double A1A2_A1A2 { get; set; }

public double A1A3_Bypass { get; set; }
public double A1A3_A1 { get; set; }
public double A1A3_A2 { get; set; }
public double A1A3_A1A2 { get; set; }

public double A2A3_Bypass { get; set; }
public double A2A3_A1 { get; set; }
public double A2A3_A2 { get; set; }
public double A2A3_A1A2 { get; set; }

public double A1A2A3_Bypass { get; set; }
public double A1A2A3_A1 { get; set; }
public double A1A2A3_A2 { get; set; }
```

```
public double A1A2A3_A1A2 { get; set; }
}

public class PortModuleUserCalNoiseConfig
{
public double Frequency { get; set; }
public SrcPort SrcPort { get; set; }
public MeasPort MeasPort { get; set; }
}

public class PortModuleNoiseCalData
{
public double ENR_Bypass { get; set; }
public double ENR_A1 { get; set; }
public double ENR_A2 { get; set; }
public double ENR_A1_A2 { get; set; }
public double ENR_A4MEAS { get; set; }
public double ENR_A4MEAS_A1 { get; set; }
public double ENR_A4MEAS_A2 { get; set; }
public double ENR_A4MEAS_A1_A2 { get; set; }

public double NoiseOn_Bypass { get; set; }
public double NoiseOn_A1 { get; set; }
public double NoiseOn_A2 { get; set; }
public double NoiseOn_A1_A2 { get; set; }
public double NoiseOn_A4MEAS { get; set; }
public double NoiseOn_A4MEAS_A1 { get; set; }
public double NoiseOn_A4MEAS_A2 { get; set; }
public double NoiseOn_A4MEAS_A1_A2 { get; set; }

public double NoiseOff_Bypass { get; set; }
public double NoiseOff_A1 { get; set; }
public double NoiseOff_A2 { get; set; }
public double NoiseOff_A1_A2 { get; set; }
public double NoiseOff_A4MEAS { get; set; }
public double NoiseOff_A4MEAS_A1 { get; set; }
public double NoiseOff_A4MEAS_A2 { get; set; }
public double NoiseOff_A4MEAS_A1_A2 { get; set; }
}

public class PortModuleUserCal
{
```

```
private bool writeToConsole = true;

public Info info = new Info();
public AmpData ampData = new AmpData();
public ExternalAttenuation attenExternal = new ExternalAttenuation();

private Dictionary<PortModuleUserCalSourceConfig, PortModuleUserCalSourceData>
sourcePortCalData = new Dictionary<PortModuleUserCalSourceConfig,
PortModuleUserCalSourceData>();
private Dictionary<PortModuleUserCalMeasureConfig, PortModuleUserCalMeasureData>
measurePortCalData = new Dictionary<PortModuleUserCalMeasureConfig,
PortModuleUserCalMeasureData>();
private Dictionary<PortModuleUserCalSourceConfig, PortModuleUserCalDirectData>
calPortCalData = new Dictionary<PortModuleUserCalSourceConfig,
PortModuleUserCalDirectData>();

private Dictionary<PortModuleUserCalSourceConfig, PortModuleUserCalSourceData>
sourcePortCalDataWithLevel = new Dictionary<PortModuleUserCalSourceConfig,
PortModuleUserCalSourceData>();
private Dictionary<PortModuleUserCalMeasureConfig, PortModuleUserCalMeasureData>
measurePortCalDataWithLevel = new Dictionary<PortModuleUserCalMeasureConfig,
PortModuleUserCalMeasureData>();
private Dictionary<PortModuleUserCalSourceConfig, PortModuleUserCalDirectData>
calPortCalDataWithLevel = new Dictionary<PortModuleUserCalSourceConfig,
PortModuleUserCalDirectData>();

private Dictionary<PortModuleUserCalSourceConfig, PortModuleUserCalSourceData>
sourcePortCalDataWithWaveform = new Dictionary<PortModuleUserCalSourceConfig,
PortModuleUserCalSourceData>();
private Dictionary<PortModuleUserCalMeasureConfig, PortModuleUserCalMeasureData>
measurePortCalDataWithWaveform = new Dictionary<PortModuleUserCalMeasureConfig,
PortModuleUserCalMeasureData>();
private Dictionary<PortModuleUserCalSourceConfig, PortModuleUserCalDirectData>
calPortCalDataWithWaveform = new Dictionary<PortModuleUserCalSourceConfig,
PortModuleUserCalDirectData>();

private Dictionary<PortModuleUserCalSourceConfig, PortModuleUserCalSourceData>
sourcePortCalDataWithWaveformAndLevel = new Dictionary<PortModuleUserCalSourceConfig,
PortModuleUserCalSourceData>();
private Dictionary<PortModuleUserCalMeasureConfig, PortModuleUserCalMeasureData>
measurePortCalDataWithWaveformAndLevel = new
Dictionary<PortModuleUserCalMeasureConfig, PortModuleUserCalMeasureData>();
```

```
private Dictionary<PortModuleUserCalSourceConfig, PortModuleUserCalDirectData>
calPortCalDataWithWaveformAndLevel = new Dictionary<PortModuleUserCalSourceConfig,
PortModuleUserCalDirectData>();

private Dictionary<PortModuleUserCalNoiseConfig, PortModuleNoiseCalData> noiseCalData =
new Dictionary<PortModuleUserCalNoiseConfig, PortModuleNoiseCalData>();

private Dictionary<WaveformKey, WaveformSettings> waveConfig = new
Dictionary<WaveformKey, WaveformSettings>();


//-------------------- Noise Calibration --------------------

public int ReadNoiseCalDataFile()
{
const string calDataFile = @"C:\MerlinTest\System\Calibration\Port
Module\APS100_RF_Measure_Port_Module_Cal_Data___201810050956.csv";

if (writeToConsole) Console.WriteLine("Loading measure port module calibration data file: {0}\n",
calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "ERROR: Port module calibration data file not found:\n" + calDataFile;
if (writeToConsole) Console.WriteLine(message);
MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Check if file is locked
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
}
catch (IOException ex)
{
MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
```

```
return -1;
}

// Read cal data from file
ImportNoiseCalData(calDataFile);

return 0;
}

public int ReadNoiseCalDataFile(string calDataFile)
{
if (writeToConsole) Console.WriteLine("Loading measure port module calibration data file: {0}\n",
calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "ERROR: Port module calibration data file not found:\n" + calDataFile;
if (writeToConsole) Console.WriteLine(message);
MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Check if file is locked
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
}
catch (IOException ex)
{
MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Read cal data from file
ImportNoiseCalData(calDataFile);

return 0;
```

```
}

private int ImportNoiseCalData(string calDataFile)
{
string line = "";
double freq = 0;

SrcPort srcPort = SrcPort.NONE;
MeasPort measPort = MeasPort.NONE;

string srcPortAlias = "";
string measPortAlias = "";

double frequency = 0;

Tuple<double, SrcPort, MeasPort> PortKey = new Tuple<double, SrcPort, MeasPort>(freq,
srcPort, measPort);

try
{
using (StreamReader stream = new StreamReader(calDataFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

//tmpString = line.Split(',')[0];
//if (tmpString.ToUpper() == "NOISE") caldata.srcSelect = SigGen.NOISE;
//else caldata.srcSelect = SigGen.ISOLATE;

tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") srcPort = SrcPort.P2;
else if (tmpString.ToUpper() == "P3") srcPort = SrcPort.P3;
else if (tmpString.ToUpper() == "P4") srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") srcPort = SrcPort.P7;
else if (tmpString.ToUpper() == "P8") srcPort = SrcPort.P8;
else if (tmpString.ToUpper() == "P9") srcPort = SrcPort.P9;
else if (tmpString.ToUpper() == "P10") srcPort = SrcPort.P10;
else if (tmpString.ToUpper() == "P11") srcPort = SrcPort.P11;
```

```
else if (tmpString.ToUpper() == "P12") srcPort = SrcPort.P12;
else if (tmpString.ToUpper() == "P13") srcPort = SrcPort.P13;
else if (tmpString.ToUpper() == "P14") srcPort = SrcPort.P14;
else if (tmpString.ToUpper() == "P15") srcPort = SrcPort.P15;
else if (tmpString.ToUpper() == "P16") srcPort = SrcPort.P16;
else if (tmpString.ToUpper() == "SG1") srcPort = SrcPort.SG1_OUT;
else if (tmpString.ToUpper() == "CAL") srcPort = SrcPort.CAL;
else srcPort = SrcPort.NONE;

srcPortAlias = line.Split(',')[2];

tmpString = line.Split(',')[3];
if (tmpString.ToUpper() == "P1") measPort = MeasPort.P1;
else if (tmpString.ToUpper() == "P2") measPort = MeasPort.P2;
else if (tmpString.ToUpper() == "P3") measPort = MeasPort.P3;
else if (tmpString.ToUpper() == "P4") measPort = MeasPort.P4;
else if (tmpString.ToUpper() == "P5") measPort = MeasPort.P5;
else if (tmpString.ToUpper() == "P6") measPort = MeasPort.P6;
else if (tmpString.ToUpper() == "P7") measPort = MeasPort.P7;
else if (tmpString.ToUpper() == "P8") measPort = MeasPort.P8;
else if (tmpString.ToUpper() == "P9") measPort = MeasPort.P9;
else if (tmpString.ToUpper() == "P10") measPort = MeasPort.P10;
else if (tmpString.ToUpper() == "P11") measPort = MeasPort.P11;
else if (tmpString.ToUpper() == "P12") measPort = MeasPort.P12;
else if (tmpString.ToUpper() == "P13") measPort = MeasPort.P13;
else if (tmpString.ToUpper() == "P14") measPort = MeasPort.P14;
else if (tmpString.ToUpper() == "P15") measPort = MeasPort.P15;
else if (tmpString.ToUpper() == "P16") measPort = MeasPort.P16;
else if (tmpString.ToUpper() == "M1") measPort = MeasPort.M1;
else if (tmpString.ToUpper() == "M2") measPort = MeasPort.M2;
else if (tmpString.ToUpper() == "M3") measPort = MeasPort.M3;
else if (tmpString.ToUpper() == "M4") measPort = MeasPort.M4;
else if (tmpString.ToUpper() == "M5") measPort = MeasPort.M5;
else if (tmpString.ToUpper() == "M6") measPort = MeasPort.M6;
else if (tmpString.ToUpper() == "M7") measPort = MeasPort.M7;
else if (tmpString.ToUpper() == "M8") measPort = MeasPort.M8;
else if (tmpString.ToUpper() == "M9") measPort = MeasPort.M9;
else if (tmpString.ToUpper() == "M10") measPort = MeasPort.M10;
else if (tmpString.ToUpper() == "M11") measPort = MeasPort.M11;
else if (tmpString.ToUpper() == "M12") measPort = MeasPort.M12;
else if (tmpString.ToUpper() == "M13") measPort = MeasPort.M13;
else if (tmpString.ToUpper() == "M14") measPort = MeasPort.M14;
```

```
else if (tmpString.ToUpper() == "M15") measPort = MeasPort.M15;
else if (tmpString.ToUpper() == "M16") measPort = MeasPort.M16;
else if (tmpString.ToUpper() == "M17") measPort = MeasPort.M17;
else if (tmpString.ToUpper() == "M18") measPort = MeasPort.M18;
else if (tmpString.ToUpper() == "M19") measPort = MeasPort.M19;
else if (tmpString.ToUpper() == "M20") measPort = MeasPort.M20;
else if (tmpString.ToUpper() == "M21") measPort = MeasPort.M21;
else if (tmpString.ToUpper() == "M22") measPort = MeasPort.M22;
else if (tmpString.ToUpper() == "M23") measPort = MeasPort.M23;
else if (tmpString.ToUpper() == "M24") measPort = MeasPort.M24;
else if (tmpString.ToUpper() == "M25") measPort = MeasPort.M25;
else if (tmpString.ToUpper() == "M26") measPort = MeasPort.M26;
else if (tmpString.ToUpper() == "MA") measPort = MeasPort.MA;
else if (tmpString.ToUpper() == "MB") measPort = MeasPort.MB;
else if (tmpString.ToUpper() == "MC") measPort = MeasPort.MC;
else if (tmpString.ToUpper() == "MD") measPort = MeasPort.MD;
else if (tmpString.ToUpper() == "NONE") measPort = MeasPort.NONE;
else measPort = MeasPort.NONE;

measPortAlias = line.Split(',')[4];

frequency = double.Parse(line.Split(',')[5]);

// Create noise cal data dictionary entry
if (!noiseCalData.Any(x => (x.Key.Frequency == freq && x.Key.SrcPort == srcPort &&
x.Key.MeasPort == measPort)))
{
PortModuleUserCalNoiseConfig portKey = new PortModuleUserCalNoiseConfig() { Frequency =
freq, SrcPort = srcPort, MeasPort = measPort };
noiseCalData.Add(portKey, new PortModuleNoiseCalData()
{
ENR_Bypass = double.Parse(line.Split(',')[6]),
ENR_A1 = double.Parse(line.Split(',')[9]),
ENR_A2 = double.Parse(line.Split(',')[12]),
ENR_A1_A2 = double.Parse(line.Split(',')[15]),

NoiseOff_Bypass = double.Parse(line.Split(',')[7]),
NoiseOff_A1 = double.Parse(line.Split(',')[10]),
NoiseOff_A2 = double.Parse(line.Split(',')[13]),
NoiseOff_A1_A2 = double.Parse(line.Split(',')[16]),

NoiseOn_Bypass = double.Parse(line.Split(',')[8]),
```

```
NoiseOn_A1 = double.Parse(line.Split(',')[11]),
NoiseOn_A2 = double.Parse(line.Split(',')[14]),
NoiseOn_A1_A2 = double.Parse(line.Split(',')[17])
});
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
return -1;
}

return 0;
}

public PortModuleNoiseCalData GetNoiseCalFactor(double freq, SrcPort srcPort, MeasPort
measPort)
{
if (noiseCalData.Any(x => (x.Key.Frequency == freq && x.Key.SrcPort == srcPort &&
x.Key.MeasPort == measPort)))
{
return noiseCalData.First(x => (x.Key.Frequency == freq && x.Key.SrcPort == srcPort &&
x.Key.MeasPort == measPort)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

public double[] GetNoiseCalFactorByAmpConfig(double freq, SrcPort srcPort, MeasPort
measPort, SrcMeasAmpConfig amp)
{
double calFactorENR = double.NaN;
double calFactorNoiseOn = double.NaN;
double calFactorNoiseOff = double.NaN;
double[] calFactors = new double[3];

PortModuleNoiseCalData calData = GetNoiseCalFactor(freq, srcPort, measPort);
```

```
switch (amp)
{
case SrcMeasAmpConfig.Bypass:
calFactorENR = calData.ENR_Bypass;
calFactorNoiseOn = calData.NoiseOn_Bypass;
calFactorNoiseOff = calData.NoiseOff_Bypass;
break;

case SrcMeasAmpConfig.A1:
calFactorENR = calData.ENR_A1;
calFactorNoiseOn = calData.NoiseOn_A1;
calFactorNoiseOff = calData.NoiseOff_A1;
break;

case SrcMeasAmpConfig.A2:
calFactorENR = calData.ENR_A2;
calFactorNoiseOn = calData.NoiseOn_A2;
calFactorNoiseOff = calData.NoiseOff_A2;
break;

case SrcMeasAmpConfig.A1_A2:
calFactorENR = calData.ENR_A1_A2;
calFactorNoiseOn = calData.NoiseOn_A1_A2;
calFactorNoiseOff = calData.NoiseOff_A1_A2;
break;

case SrcMeasAmpConfig.A4_MEAS:
calFactorENR = calData.ENR_A4MEAS;
calFactorNoiseOn = calData.NoiseOn_A4MEAS;
calFactorNoiseOff = calData.NoiseOff_A4MEAS;
break;

case SrcMeasAmpConfig.A4_MEAS_A1:
calFactorENR = calData.ENR_A4MEAS_A1;
calFactorNoiseOn = calData.NoiseOn_A4MEAS_A1;
calFactorNoiseOff = calData.NoiseOff_A4MEAS_A1;
break;

case SrcMeasAmpConfig.A4_MEAS_A2:
calFactorENR = calData.ENR_A4MEAS_A2;
calFactorNoiseOn = calData.NoiseOn_A4MEAS_A2;
```

```
calFactorNoiseOff = calData.NoiseOff_A4MEAS_A2;
break;

case SrcMeasAmpConfig.A4_MEAS_A1_A2:
calFactorENR = calData.ENR_A4MEAS_A1_A2;
calFactorNoiseOn = calData.NoiseOn_A4MEAS_A1_A2;
calFactorNoiseOff = calData.NoiseOff_A4MEAS_A1_A2;
break;
}

calFactors[0] = calFactorENR;
calFactors[1] = calFactorNoiseOn;
calFactors[2] = calFactorNoiseOff;

return calFactors;

}

//---------------------- RF Calibration ----------------------

public int ReadCalDataFile()
{
const string calDataFile = @"C:\MerlinTest\System\Calibration\Port
Module\APS100_RF_Measure_Port_Module_Cal_Data___201810050956.csv";

if (writeToConsole) Console.WriteLine("Loading measure port module calibration data file: {0}\n",
calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "ERROR: Port module calibration data file not found:\n" + calDataFile;
if (writeToConsole) Console.WriteLine(message);
MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Check if file is locked
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
```

```
FileShare.None);
fs.Close();
}
catch (IOException ex)
{
MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Read cal data from file
ImportCalData(calDataFile);

return 0;
}

public int ReadCalDataFile(string calDataFile)
{
if (writeToConsole) Console.WriteLine("Loading port module user calibration data file: {0}\n",
calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "ERROR: Port module calibration data file not found:\n" + calDataFile;
if (writeToConsole) Console.WriteLine(message);
MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Check if file is locked
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
}
catch (IOException ex)
{
MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
```

```
return -1;
}

// Read cal data from file
ImportCalData(calDataFile);

return 0;
}

private int ImportCalConfig(string calConfigFile)
{
string line = "";
Info info = new Info();
CalSettings CalSelect = new CalSettings();
ExternalAttenuation attenExternal = new ExternalAttenuation();
CalDataPortModule caldata = new CalDataPortModule();

try
{
using (StreamReader stream = new StreamReader(calConfigFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
if (tmpString.ToUpper() == "REVISION:")
{
info.Revision = line.Split(',')[1];
}
if (tmpString.ToUpper() == "TEST PROGRAM:")
{
info.Program = line.Split(',')[1];
}
if (tmpString.ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = line.Split(',')[1];
}
if (tmpString.ToUpper() == "COMMENT:")
```

```
{
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "USE EXTERNAL POWER METER:")
{
CalSelect.PowerMeterAvailable = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE RF SOURCE/MEASURE:")
{
CalSelect.SourceMeasurePath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE INTERNAL DIRECT PATH:")
{
CalSelect.InternalMatrixPath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE NOISE SOURCE:")
{
CalSelect.NoiseSource = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE DC SENSE:")
{
CalSelect.DCSense = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE VNA:")
{
CalSelect.VNA = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "NOISE BANDWIDTH (HZ):")
{
CalSelect.NoiseBandwidth = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "SOURCE PATH ATTENUATORS:")
{
const int pathCount = 17;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenExternal.srcAttenPortModule[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH ATTENUATORS:")
{
```

```
const int pathCount = 30;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenExternal.measAttenPortModule[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG2" || tmpString.ToUpper() ==
"NOISE")
{
tmpString = line.Split(',')[0];
if (tmpString.ToUpper() == "SG1") caldata.srcSelect = SigGen.SG1;
else if (tmpString.ToUpper() == "SG2") caldata.srcSelect = SigGen.SG2;
else if (tmpString.ToUpper() == "NOISE") caldata.srcSelect = SigGen.NOISE;

tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") caldata.srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") caldata.srcPort = SrcPort.P2;
else if (tmpString.ToUpper() == "P3") caldata.srcPort = SrcPort.P3;
else if (tmpString.ToUpper() == "P4") caldata.srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") caldata.srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") caldata.srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") caldata.srcPort = SrcPort.P7;
else if (tmpString.ToUpper() == "P8") caldata.srcPort = SrcPort.P8;
else if (tmpString.ToUpper() == "P9") caldata.srcPort = SrcPort.P9;
else if (tmpString.ToUpper() == "P10") caldata.srcPort = SrcPort.P10;
else if (tmpString.ToUpper() == "P11") caldata.srcPort = SrcPort.P11;
else if (tmpString.ToUpper() == "P12") caldata.srcPort = SrcPort.P12;
else if (tmpString.ToUpper() == "P13") caldata.srcPort = SrcPort.P13;
else if (tmpString.ToUpper() == "P14") caldata.srcPort = SrcPort.P14;
else if (tmpString.ToUpper() == "P15") caldata.srcPort = SrcPort.P15;
else if (tmpString.ToUpper() == "P16") caldata.srcPort = SrcPort.P16;
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG1_OUT") caldata.srcPort =
SrcPort.SG1_OUT;
else if (tmpString.ToUpper() == "CAL") caldata.srcPort = SrcPort.CAL;
else caldata.srcPort = SrcPort.NONE;

caldata.srcPortAlias = line.Split(',')[2];

tmpString = line.Split(',')[3];
if (tmpString.ToUpper() == "P1") caldata.measPort = MeasPort.P1;
else if (tmpString.ToUpper() == "P2") caldata.measPort = MeasPort.P2;
```

```
else if (tmpString.ToUpper() == "P3") caldata.measPort = MeasPort.P3;
else if (tmpString.ToUpper() == "P4") caldata.measPort = MeasPort.P4;
else if (tmpString.ToUpper() == "P5") caldata.measPort = MeasPort.P5;
else if (tmpString.ToUpper() == "P6") caldata.measPort = MeasPort.P6;
else if (tmpString.ToUpper() == "P7") caldata.measPort = MeasPort.P7;
else if (tmpString.ToUpper() == "P8") caldata.measPort = MeasPort.P8;
else if (tmpString.ToUpper() == "P9") caldata.measPort = MeasPort.P9;
else if (tmpString.ToUpper() == "P10") caldata.measPort = MeasPort.P10;
else if (tmpString.ToUpper() == "P11") caldata.measPort = MeasPort.P11;
else if (tmpString.ToUpper() == "P12") caldata.measPort = MeasPort.P12;
else if (tmpString.ToUpper() == "P13") caldata.measPort = MeasPort.P13;
else if (tmpString.ToUpper() == "P14") caldata.measPort = MeasPort.P14;
else if (tmpString.ToUpper() == "P15") caldata.measPort = MeasPort.P15;
else if (tmpString.ToUpper() == "P16") caldata.measPort = MeasPort.P16;
else if (tmpString.ToUpper() == "P1_FWD") caldata.measPort = MeasPort.P1_Fwd;
else if (tmpString.ToUpper() == "P2_FWD") caldata.measPort = MeasPort.P2_Fwd;
else if (tmpString.ToUpper() == "P3_FWD") caldata.measPort = MeasPort.P3_Fwd;
else if (tmpString.ToUpper() == "P4_FWD") caldata.measPort = MeasPort.P4_Fwd;
else if (tmpString.ToUpper() == "P5_FWD") caldata.measPort = MeasPort.P5_Fwd;
else if (tmpString.ToUpper() == "P6_FWD") caldata.measPort = MeasPort.P6_Fwd;
else if (tmpString.ToUpper() == "P7_FWD") caldata.measPort = MeasPort.P7_Fwd;
else if (tmpString.ToUpper() == "P8_FWD") caldata.measPort = MeasPort.P8_Fwd;
else if (tmpString.ToUpper() == "P1_REV") caldata.measPort = MeasPort.P1_Rev;
else if (tmpString.ToUpper() == "P2_REV") caldata.measPort = MeasPort.P2_Rev;
else if (tmpString.ToUpper() == "P3_REV") caldata.measPort = MeasPort.P3_Rev;
else if (tmpString.ToUpper() == "P4_REV") caldata.measPort = MeasPort.P4_Rev;
else if (tmpString.ToUpper() == "P5_REV") caldata.measPort = MeasPort.P5_Rev;
else if (tmpString.ToUpper() == "P6_REV") caldata.measPort = MeasPort.P6_Rev;
else if (tmpString.ToUpper() == "P7_REV") caldata.measPort = MeasPort.P7_Rev;
else if (tmpString.ToUpper() == "P8_REV") caldata.measPort = MeasPort.P8_Rev;
else if (tmpString.ToUpper() == "M1") caldata.measPort = MeasPort.M1;
else if (tmpString.ToUpper() == "M2") caldata.measPort = MeasPort.M2;
else if (tmpString.ToUpper() == "M3") caldata.measPort = MeasPort.M3;
else if (tmpString.ToUpper() == "M4") caldata.measPort = MeasPort.M4;
else if (tmpString.ToUpper() == "M5") caldata.measPort = MeasPort.M5;
else if (tmpString.ToUpper() == "M6") caldata.measPort = MeasPort.M6;
else if (tmpString.ToUpper() == "M7") caldata.measPort = MeasPort.M7;
else if (tmpString.ToUpper() == "M8") caldata.measPort = MeasPort.M8;
else if (tmpString.ToUpper() == "M9") caldata.measPort = MeasPort.M9;
else if (tmpString.ToUpper() == "M10") caldata.measPort = MeasPort.M10;
else if (tmpString.ToUpper() == "M11") caldata.measPort = MeasPort.M11;
else if (tmpString.ToUpper() == "M12") caldata.measPort = MeasPort.M12;
```

```
else if (tmpString.ToUpper() == "M13") caldata.measPort = MeasPort.M13;
else if (tmpString.ToUpper() == "M14") caldata.measPort = MeasPort.M14;
else if (tmpString.ToUpper() == "M15") caldata.measPort = MeasPort.M15;
else if (tmpString.ToUpper() == "M16") caldata.measPort = MeasPort.M16;
else if (tmpString.ToUpper() == "M17") caldata.measPort = MeasPort.M17;
else if (tmpString.ToUpper() == "M18") caldata.measPort = MeasPort.M18;
else if (tmpString.ToUpper() == "M19") caldata.measPort = MeasPort.M19;
else if (tmpString.ToUpper() == "M20") caldata.measPort = MeasPort.M20;
else if (tmpString.ToUpper() == "M21") caldata.measPort = MeasPort.M21;
else if (tmpString.ToUpper() == "M22") caldata.measPort = MeasPort.M22;
else if (tmpString.ToUpper() == "M23") caldata.measPort = MeasPort.M23;
else if (tmpString.ToUpper() == "M24") caldata.measPort = MeasPort.M24;
else if (tmpString.ToUpper() == "M25") caldata.measPort = MeasPort.M25;
else if (tmpString.ToUpper() == "M26") caldata.measPort = MeasPort.M26;
else if (tmpString.ToUpper() == "MA") caldata.measPort = MeasPort.MA;
else if (tmpString.ToUpper() == "MB") caldata.measPort = MeasPort.MB;
else if (tmpString.ToUpper() == "MC") caldata.measPort = MeasPort.MC;
else if (tmpString.ToUpper() == "MD") caldata.measPort = MeasPort.MD;
else if (tmpString.ToUpper() == "CAL") caldata.measPort = MeasPort.CAL;
else if (tmpString.ToUpper() == "NONE") caldata.measPort = MeasPort.NONE;
else caldata.measPort = MeasPort.NONE;

caldata.measPortAlias = line.Split(',')[4];
caldata.srcFreq = double.Parse(line.Split(',')[5]);
caldata.srcLevel = double.Parse(line.Split(',')[6]);

tmpString = line.Split(',')[7];
if (tmpString.ToUpper() == "BYPASS") caldata.measFilter = MeasFilt.Bypass;
else if (tmpString.ToUpper() == "ATTEN") caldata.measFilter = MeasFilt.Atten;
else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1") caldata.measFilter =
MeasFilt.FILT1;
else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2") caldata.measFilter =
MeasFilt.FILT2;
else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3") caldata.measFilter =
MeasFilt.FILT3;
else caldata.measFilter = MeasFilt.Bypass; // Default

caldata.measAtten = double.Parse(line.Split(',')[8]);
caldata.modulationType = line.Split(',')[9];
caldata.modulationFile = line.Split(',')[10];
caldata.dutyCycle = double.Parse(line.Split(',')[11]);
caldata.comment = line.Split(',')[12];
```

```
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
return -1;
}

return 0;
}

private int ImportCalData(string calDataFile)
{
int index = 0;
string line = "";
SigGen srcSelect = SigGen.ISOLATE;
SrcPort srcPort = SrcPort.NONE;
MeasPort measPort = MeasPort.NONE;
MeasFilt measFilter = MeasFilt.NONE;
string srcPortAlias = "";
string measPortAlias = "";
double srcFreq = 0;
double srcLevel = 0;
double measAtten = 0;
string modulationType = "";
string modulationFile = "";
double bandwidth = 0;
double dutyCycle = 100;
int marker = -1;
int analysisSlot = 0;
string comment = "";
double[] srcCalFactor = new double[8];
double[] measCalFactor = new double[8];
double[] calCalFactor = new double[4];

CalSettings CalSelect = new CalSettings();
List<CalDataPortModule> waveData = new List<CalDataPortModule>();

try
{
```

```
using (StreamReader stream = new StreamReader(calDataFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
if (tmpString.ToUpper() == "REVISION:")
{
info.Revision = line.Split(',')[1];
}
if (tmpString.ToUpper() == "TEST PROGRAM:")
{
info.Program = line.Split(',')[1];
}
if (tmpString.ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = line.Split(',')[1];
}
if (tmpString.ToUpper() == "COMMENT:")
{
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "USE EXTERNAL POWER METER:")
{
CalSelect.PowerMeterAvailable = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE RF SOURCE/MEASURE:")
{
CalSelect.SourceMeasurePath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE INTERNAL DIRECT PATH:")
{
CalSelect.InternalMatrixPath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE NOISE SOURCE:")
{
CalSelect.NoiseSource = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
```

```
else if (tmpString.ToUpper() == "CALIBRATE VNA:")
{
CalSelect.VNA = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE DC SENSE:")
{
CalSelect.DCSense = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "NOISE BANDWIDTH (HZ):")
{
CalSelect.NoiseBandwidth = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "SOURCE PATH ATTENUATORS:")
{
const int pathCount = 17;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenExternal.srcAttenPortModule[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH ATTENUATORS:")
{
const int pathCount = 30;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenExternal.measAttenPortModule[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG2" || tmpString.ToUpper() ==
"NOISE")
{
CalDataPortModule caldata = new CalDataPortModule();

tmpString = line.Split(',')[0];
if (tmpString.ToUpper() == "SG1") srcSelect = SigGen.SG1;
else if (tmpString.ToUpper() == "SG2") srcSelect = SigGen.SG2;

tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") srcPort = SrcPort.P2;
```

```
else if (tmpString.ToUpper() == "P3") srcPort = SrcPort.P3;
else if (tmpString.ToUpper() == "P4") srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") srcPort = SrcPort.P7;
else if (tmpString.ToUpper() == "P8") srcPort = SrcPort.P8;
else if (tmpString.ToUpper() == "P9") srcPort = SrcPort.P9;
else if (tmpString.ToUpper() == "P10") srcPort = SrcPort.P10;
else if (tmpString.ToUpper() == "P11") srcPort = SrcPort.P11;
else if (tmpString.ToUpper() == "P12") srcPort = SrcPort.P12;
else if (tmpString.ToUpper() == "P13") srcPort = SrcPort.P13;
else if (tmpString.ToUpper() == "P14") srcPort = SrcPort.P14;
else if (tmpString.ToUpper() == "P15") srcPort = SrcPort.P15;
else if (tmpString.ToUpper() == "P16") srcPort = SrcPort.P16;
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG1_OUT") srcPort =
SrcPort.SG1_OUT;
else if (tmpString.ToUpper() == "CAL") srcPort = SrcPort.CAL;
else if (tmpString.ToUpper() == "NONE") srcPort = SrcPort.NONE;
else srcPort = SrcPort.NONE;

srcPortAlias = line.Split(',')[2];

tmpString = line.Split(',')[3];
if (tmpString.ToUpper() == "P1") measPort = MeasPort.P1;
else if (tmpString.ToUpper() == "P2") measPort = MeasPort.P2;
else if (tmpString.ToUpper() == "P3") measPort = MeasPort.P3;
else if (tmpString.ToUpper() == "P4") measPort = MeasPort.P4;
else if (tmpString.ToUpper() == "P5") measPort = MeasPort.P5;
else if (tmpString.ToUpper() == "P6") measPort = MeasPort.P6;
else if (tmpString.ToUpper() == "P7") measPort = MeasPort.P7;
else if (tmpString.ToUpper() == "P8") measPort = MeasPort.P8;
else if (tmpString.ToUpper() == "P9") measPort = MeasPort.P9;
else if (tmpString.ToUpper() == "P10") measPort = MeasPort.P10;
else if (tmpString.ToUpper() == "P11") measPort = MeasPort.P11;
else if (tmpString.ToUpper() == "P12") measPort = MeasPort.P12;
else if (tmpString.ToUpper() == "P13") measPort = MeasPort.P13;
else if (tmpString.ToUpper() == "P14") measPort = MeasPort.P14;
else if (tmpString.ToUpper() == "P15") measPort = MeasPort.P15;
else if (tmpString.ToUpper() == "P16") measPort = MeasPort.P16;
else if (tmpString.ToUpper() == "P1_FWD") measPort = MeasPort.P1_Fwd;
else if (tmpString.ToUpper() == "P2_FWD") measPort = MeasPort.P2_Fwd;
else if (tmpString.ToUpper() == "P3_FWD") measPort = MeasPort.P3_Fwd;
```

```
else if (tmpString.ToUpper() == "P4_FWD") measPort = MeasPort.P4_Fwd;
else if (tmpString.ToUpper() == "P5_FWD") measPort = MeasPort.P5_Fwd;
else if (tmpString.ToUpper() == "P6_FWD") measPort = MeasPort.P6_Fwd;
else if (tmpString.ToUpper() == "P7_FWD") measPort = MeasPort.P7_Fwd;
else if (tmpString.ToUpper() == "P8_FWD") measPort = MeasPort.P8_Fwd;
else if (tmpString.ToUpper() == "P1_REV") measPort = MeasPort.P1_Rev;
else if (tmpString.ToUpper() == "P2_REV") measPort = MeasPort.P2_Rev;
else if (tmpString.ToUpper() == "P3_REV") measPort = MeasPort.P3_Rev;
else if (tmpString.ToUpper() == "P4_REV") measPort = MeasPort.P4_Rev;
else if (tmpString.ToUpper() == "P5_REV") measPort = MeasPort.P5_Rev;
else if (tmpString.ToUpper() == "P6_REV") measPort = MeasPort.P6_Rev;
else if (tmpString.ToUpper() == "P7_REV") measPort = MeasPort.P7_Rev;
else if (tmpString.ToUpper() == "P8_REV") measPort = MeasPort.P8_Rev;
else if (tmpString.ToUpper() == "M1") measPort = MeasPort.M1;
else if (tmpString.ToUpper() == "M2") measPort = MeasPort.M2;
else if (tmpString.ToUpper() == "M3") measPort = MeasPort.M3;
else if (tmpString.ToUpper() == "M4") measPort = MeasPort.M4;
else if (tmpString.ToUpper() == "M5") measPort = MeasPort.M5;
else if (tmpString.ToUpper() == "M6") measPort = MeasPort.M6;
else if (tmpString.ToUpper() == "M7") measPort = MeasPort.M7;
else if (tmpString.ToUpper() == "M8") measPort = MeasPort.M8;
else if (tmpString.ToUpper() == "M9") measPort = MeasPort.M9;
else if (tmpString.ToUpper() == "M10") measPort = MeasPort.M10;
else if (tmpString.ToUpper() == "M11") measPort = MeasPort.M11;
else if (tmpString.ToUpper() == "M12") measPort = MeasPort.M12;
else if (tmpString.ToUpper() == "M13") measPort = MeasPort.M13;
else if (tmpString.ToUpper() == "M14") measPort = MeasPort.M14;
else if (tmpString.ToUpper() == "M15") measPort = MeasPort.M15;
else if (tmpString.ToUpper() == "M16") measPort = MeasPort.M16;
else if (tmpString.ToUpper() == "M17") measPort = MeasPort.M17;
else if (tmpString.ToUpper() == "M18") measPort = MeasPort.M18;
else if (tmpString.ToUpper() == "M19") measPort = MeasPort.M19;
else if (tmpString.ToUpper() == "M20") measPort = MeasPort.M20;
else if (tmpString.ToUpper() == "M21") measPort = MeasPort.M21;
else if (tmpString.ToUpper() == "M22") measPort = MeasPort.M22;
else if (tmpString.ToUpper() == "M23") measPort = MeasPort.M23;
else if (tmpString.ToUpper() == "M24") measPort = MeasPort.M24;
else if (tmpString.ToUpper() == "M25") measPort = MeasPort.M25;
else if (tmpString.ToUpper() == "M26") measPort = MeasPort.M26;
else if (tmpString.ToUpper() == "MA") measPort = MeasPort.MA;
else if (tmpString.ToUpper() == "MB") measPort = MeasPort.MB;
else if (tmpString.ToUpper() == "MC") measPort = MeasPort.MC;
```

```
else if (tmpString.ToUpper() == "MD") measPort = MeasPort.MD;
else if (tmpString.ToUpper() == "CAL") measPort = MeasPort.CAL;
else if (tmpString.ToUpper() == "NONE") measPort = MeasPort.NONE;
else measPort = MeasPort.NONE;

measPortAlias = line.Split(',')[4];
srcFreq = double.Parse(line.Split(',')[5]);
srcLevel = double.Parse(line.Split(',')[6]);

tmpString = line.Split(',')[7];
if (tmpString.ToUpper() == "BYPASS") measFilter = MeasFilt.Bypass;
else if (tmpString.ToUpper() == "ATTEN") measFilter = MeasFilt.Atten;
else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1") measFilter =
MeasFilt.FILT1;
else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2") measFilter =
MeasFilt.FILT2;
else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3") measFilter =
MeasFilt.FILT3;
else measFilter = MeasFilt.Bypass; // Default

measAtten = double.Parse(line.Split(',')[8]);
modulationType = line.Split(',')[9];
if (info.CalVersion.ToUpper() == "V5")
{
modulationFile = Path.GetFileNameWithoutExtension(line.Split(',')[10]);

caldata.modulationFile = modulationFile;
caldata.modulationType = modulationType;
caldata.WaveConfig = new WaveformSettings();
waveData.Add(caldata);

if (modulationFile.ToUpper() != "" && modulationFile.ToUpper() != "NONE") modulationFile +=
".aiq"; // Added for 5G compatibility
//tmpString = line.Split(',')[11];
//if (tmpString != "" && tmpString.ToUpper() != "EXT") marker = int.Parse(line.Split(',')[11]);
//else if (tmpString.ToUpper() == "EXT") marker = 0;
//tmpString = line.Split(',')[12];
//if (tmpString.ToUpper() != String.Empty) analysisSlot = int.Parse(line.Split(',')[12]);
//tmpString = line.Split(',')[13];
//if (tmpString.ToUpper() != String.Empty) bandwidth = int.Parse(line.Split(',')[13]);
//dutyCycle = double.Parse(line.Split(',')[14]);
comment = line.Split(',')[15];
```

```
index = 16;
}
else
{
modulationFile = line.Split(',')[10];
dutyCycle = double.Parse(line.Split(',')[11]);
comment = line.Split(',')[12];
index = 13;
}

if (!sourcePortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulationType)))
{
PortModuleUserCalSourceConfig sourceKey = new PortModuleUserCalSourceConfig() {
Frequency = srcFreq, SigGen = srcSelect, SrcPort = srcPort, Modulation = modulationType };
sourcePortCalData.Add(sourceKey, new PortModuleUserCalSourceData()
{
Bypass = double.Parse(line.Split(',')[index++]),
A1 = double.Parse(line.Split(',')[index++]),
A2 = double.Parse(line.Split(',')[index++]),
A3 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A1_A3 = double.Parse(line.Split(',')[index++]),
A2_A3 = double.Parse(line.Split(',')[index++]),
A1_A2_A3 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 16;
else index = 13;

if (!sourcePortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulationType && x.Key.Level == srcLevel)))
{
PortModuleUserCalSourceConfig sourceKey = new PortModuleUserCalSourceConfig() {
Frequency = srcFreq, SigGen = srcSelect, SrcPort = srcPort, Modulation = modulationType, Level
= srcLevel };
sourcePortCalDataWithLevel.Add(sourceKey, new PortModuleUserCalSourceData()
{
Bypass = double.Parse(line.Split(',')[index++]),
A1 = double.Parse(line.Split(',')[index++]),
```

```
A2 = double.Parse(line.Split(',')[index++]),
A3 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A1_A3 = double.Parse(line.Split(',')[index++]),
A2_A3 = double.Parse(line.Split(',')[index++]),
A1_A2_A3 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 16;
else index = 13;

if (!sourcePortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulationType && x.Key.Waveform ==
modulationFile)))
{
PortModuleUserCalSourceConfig sourceKey = new PortModuleUserCalSourceConfig() {
Frequency = srcFreq, SigGen = srcSelect, SrcPort = srcPort, Modulation = modulationType,
Waveform = modulationFile };
sourcePortCalData.Add(sourceKey, new PortModuleUserCalSourceData()
{
Bypass = double.Parse(line.Split(',')[index++]),
A1 = double.Parse(line.Split(',')[index++]),
A2 = double.Parse(line.Split(',')[index++]),
A3 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A1_A3 = double.Parse(line.Split(',')[index++]),
A2_A3 = double.Parse(line.Split(',')[index++]),
A1_A2_A3 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 16;
else index = 13;

if (!sourcePortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulationType && x.Key.Waveform ==
modulationFile && x.Key.Level == srcLevel)))
{
PortModuleUserCalSourceConfig sourceKey = new PortModuleUserCalSourceConfig() {
Frequency = srcFreq, SigGen = srcSelect, SrcPort = srcPort, Modulation = modulationType,
Waveform = modulationFile, Level = srcLevel };
```

```
sourcePortCalDataWithLevel.Add(sourceKey, new PortModuleUserCalSourceData()
{
Bypass = double.Parse(line.Split(',')[index++]),
A1 = double.Parse(line.Split(',')[index++]),
A2 = double.Parse(line.Split(',')[index++]),
A3 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A1_A3 = double.Parse(line.Split(',')[index++]),
A2_A3 = double.Parse(line.Split(',')[index++]),
A1_A2_A3 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 24;
else index = 21;

if (!measurePortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect
&& x.Key.MeasPort == measPort && x.Key.Attenuator == measFilter && x.Key.Modulation ==
modulationType)))
{
PortModuleUserCalMeasureConfig measureKey = new PortModuleUserCalMeasureConfig() {
Frequency = srcFreq, SigGen = srcSelect, MeasPort = measPort, Attenuator = measFilter,
Modulation = modulationType };
measurePortCalData.Add(measureKey, new PortModuleUserCalMeasureData()
{
Bypass = double.Parse(line.Split(',')[index++]),
A1 = double.Parse(line.Split(',')[index++]),
A2 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A4_MEAS = double.Parse(line.Split(',')[index++]),
A4_MEAS_A1 = double.Parse(line.Split(',')[index++]),
A4_MEAS_A2 = double.Parse(line.Split(',')[index++]),
A4_MEAS_A1_A2 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 24;
else index = 21;

if (!measurePortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect
&& x.Key.MeasPort == measPort && x.Key.Attenuator == measFilter && x.Key.Modulation ==
modulationType && x.Key.Level == srcLevel)))
```

```
{
PortModuleUserCalMeasureConfig measureKey = new PortModuleUserCalMeasureConfig() {
Frequency = srcFreq, SigGen = srcSelect, MeasPort = measPort, Attenuator = measFilter,
Modulation = modulationType, Level = srcLevel };
measurePortCalDataWithLevel.Add(measureKey, new PortModuleUserCalMeasureData()
{
Bypass = double.Parse(line.Split(',')[index++]),
A1 = double.Parse(line.Split(',')[index++]),
A2 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A4_MEAS = double.Parse(line.Split(',')[index++]),
A4_MEAS_A1 = double.Parse(line.Split(',')[index++]),
A4_MEAS_A2 = double.Parse(line.Split(',')[index++]),
A4_MEAS_A1_A2 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 24;
else index = 21;

if (!measurePortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect
&& x.Key.MeasPort == measPort && x.Key.Attenuator == measFilter && x.Key.Modulation ==
modulationType && x.Key.Waveform == modulationFile)))
{
PortModuleUserCalMeasureConfig measureKey = new PortModuleUserCalMeasureConfig() {
Frequency = srcFreq, SigGen = srcSelect, MeasPort = measPort, Attenuator = measFilter,
Modulation = modulationType, Waveform = modulationFile };
measurePortCalData.Add(measureKey, new PortModuleUserCalMeasureData()
{
Bypass = double.Parse(line.Split(',')[index++]),
A1 = double.Parse(line.Split(',')[index++]),
A2 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A4_MEAS = double.Parse(line.Split(',')[index++]),
A4_MEAS_A1 = double.Parse(line.Split(',')[index++]),
A4_MEAS_A2 = double.Parse(line.Split(',')[index++]),
A4_MEAS_A1_A2 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 24;
else index = 21;
```

```
if (!measurePortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect
&& x.Key.MeasPort == measPort && x.Key.Attenuator == measFilter && x.Key.Modulation ==
modulationType && x.Key.Waveform == modulationFile && x.Key.Level == srcLevel)))
{
PortModuleUserCalMeasureConfig measureKey = new PortModuleUserCalMeasureConfig() {
Frequency = srcFreq, SigGen = srcSelect, MeasPort = measPort, Attenuator = measFilter,
Modulation = modulationType, Waveform = modulationFile, Level = srcLevel };
measurePortCalDataWithLevel.Add(measureKey, new PortModuleUserCalMeasureData()
{
Bypass = double.Parse(line.Split(',')[index++]),
A1 = double.Parse(line.Split(',')[index++]),
A2 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A4_MEAS = double.Parse(line.Split(',')[index++]),
A4_MEAS_A1 = double.Parse(line.Split(',')[index++]),
A4_MEAS_A2 = double.Parse(line.Split(',')[index++]),
A4_MEAS_A1_A2 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 32;
else index = 29;

if (!calPortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulationType)))
{
PortModuleUserCalSourceConfig calKey = new PortModuleUserCalSourceConfig() { Frequency =
srcFreq, SigGen = srcSelect, SrcPort = srcPort, Modulation = modulationType };
calPortCalData.Add(calKey, new PortModuleUserCalDirectData()
{
Bypass = double.Parse(line.Split(',')[index++]),
Bypass_A1 = double.Parse(line.Split(',')[index++]),
Bypass_A2 = double.Parse(line.Split(',')[index++]),
Bypass_A1A2 = double.Parse(line.Split(',')[index++]),

A1_Bypass = double.Parse(line.Split(',')[index++]),
A1_A1 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A1_A1A2 = double.Parse(line.Split(',')[index++]),

A2_Bypass = double.Parse(line.Split(',')[index++]),
```

```
A2_A1 = double.Parse(line.Split(',')[index++]),
A2_A2 = double.Parse(line.Split(',')[index++]),
A2_A1A2 = double.Parse(line.Split(',')[index++]),

A3_Bypass = double.Parse(line.Split(',')[index++]),
A3_A1 = double.Parse(line.Split(',')[index++]),
A3_A2 = double.Parse(line.Split(',')[index++]),
A3_A1A2 = double.Parse(line.Split(',')[index++]),

A1A2_Bypass = double.Parse(line.Split(',')[index++]),
A1A2_A1 = double.Parse(line.Split(',')[index++]),
A1A2_A2 = double.Parse(line.Split(',')[index++]),
A1A2_A1A2 = double.Parse(line.Split(',')[index++]),

A1A3_Bypass = double.Parse(line.Split(',')[index++]),
A1A3_A1 = double.Parse(line.Split(',')[index++]),
A1A3_A2 = double.Parse(line.Split(',')[index++]),
A1A3_A1A2 = double.Parse(line.Split(',')[index++]),

A2A3_Bypass = double.Parse(line.Split(',')[index++]),
A2A3_A1 = double.Parse(line.Split(',')[index++]),
A2A3_A2 = double.Parse(line.Split(',')[index++]),
A2A3_A1A2 = double.Parse(line.Split(',')[index++]),

A1A2A3_Bypass = double.Parse(line.Split(',')[index++]),
A1A2A3_A1 = double.Parse(line.Split(',')[index++]),
A1A2A3_A2 = double.Parse(line.Split(',')[index++]),
A1A2A3_A1A2 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 32;
else index = 29;

if (!calPortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulationType && x.Key.Level == srcLevel)))
{
PortModuleUserCalSourceConfig calKey = new PortModuleUserCalSourceConfig() { Frequency =
srcFreq, SigGen = srcSelect, SrcPort = srcPort, Modulation = modulationType, Level = srcLevel };
calPortCalDataWithLevel.Add(calKey, new PortModuleUserCalDirectData()
{
Bypass = double.Parse(line.Split(',')[index++]),
```

```
Bypass_A1 = double.Parse(line.Split(',')[index++]),
Bypass_A2 = double.Parse(line.Split(',')[index++]),
Bypass_A1A2 = double.Parse(line.Split(',')[index++]),

A1_Bypass = double.Parse(line.Split(',')[index++]),
A1_A1 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A1_A1A2 = double.Parse(line.Split(',')[index++]),

A2_Bypass = double.Parse(line.Split(',')[index++]),
A2_A1 = double.Parse(line.Split(',')[index++]),
A2_A2 = double.Parse(line.Split(',')[index++]),
A2_A1A2 = double.Parse(line.Split(',')[index++]),

A3_Bypass = double.Parse(line.Split(',')[index++]),
A3_A1 = double.Parse(line.Split(',')[index++]),
A3_A2 = double.Parse(line.Split(',')[index++]),
A3_A1A2 = double.Parse(line.Split(',')[index++]),

A1A2_Bypass = double.Parse(line.Split(',')[index++]),
A1A2_A1 = double.Parse(line.Split(',')[index++]),
A1A2_A2 = double.Parse(line.Split(',')[index++]),
A1A2_A1A2 = double.Parse(line.Split(',')[index++]),

A1A3_Bypass = double.Parse(line.Split(',')[index++]),
A1A3_A1 = double.Parse(line.Split(',')[index++]),
A1A3_A2 = double.Parse(line.Split(',')[index++]),
A1A3_A1A2 = double.Parse(line.Split(',')[index++]),

A2A3_Bypass = double.Parse(line.Split(',')[index++]),
A2A3_A1 = double.Parse(line.Split(',')[index++]),
A2A3_A2 = double.Parse(line.Split(',')[index++]),
A2A3_A1A2 = double.Parse(line.Split(',')[index++]),

A1A2A3_Bypass = double.Parse(line.Split(',')[index++]),
A1A2A3_A1 = double.Parse(line.Split(',')[index++]),
A1A2A3_A2 = double.Parse(line.Split(',')[index++]),
A1A2A3_A1A2 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 32;
```

```
else index = 29;

if (!calPortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulationType && x.Key.Waveform ==
modulationFile)))
{
PortModuleUserCalSourceConfig calKey = new PortModuleUserCalSourceConfig() { Frequency =
srcFreq, SigGen = srcSelect, SrcPort = srcPort, Modulation = modulationType, Waveform =
modulationFile };
calPortCalData.Add(calKey, new PortModuleUserCalDirectData()
{
Bypass = double.Parse(line.Split(',')[index++]),
Bypass_A1 = double.Parse(line.Split(',')[index++]),
Bypass_A2 = double.Parse(line.Split(',')[index++]),
Bypass_A1A2 = double.Parse(line.Split(',')[index++]),

A1_Bypass = double.Parse(line.Split(',')[index++]),
A1_A1 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A1_A1A2 = double.Parse(line.Split(',')[index++]),

A2_Bypass = double.Parse(line.Split(',')[index++]),
A2_A1 = double.Parse(line.Split(',')[index++]),
A2_A2 = double.Parse(line.Split(',')[index++]),
A2_A1A2 = double.Parse(line.Split(',')[index++]),

A3_Bypass = double.Parse(line.Split(',')[index++]),
A3_A1 = double.Parse(line.Split(',')[index++]),
A3_A2 = double.Parse(line.Split(',')[index++]),
A3_A1A2 = double.Parse(line.Split(',')[index++]),

A1A2_Bypass = double.Parse(line.Split(',')[index++]),
A1A2_A1 = double.Parse(line.Split(',')[index++]),
A1A2_A2 = double.Parse(line.Split(',')[index++]),
A1A2_A1A2 = double.Parse(line.Split(',')[index++]),

A1A3_Bypass = double.Parse(line.Split(',')[index++]),
A1A3_A1 = double.Parse(line.Split(',')[index++]),
A1A3_A2 = double.Parse(line.Split(',')[index++]),
A1A3_A1A2 = double.Parse(line.Split(',')[index++]),

A2A3_Bypass = double.Parse(line.Split(',')[index++]),
```

```
A2A3_A1 = double.Parse(line.Split(',')[index++]),
A2A3_A2 = double.Parse(line.Split(',')[index++]),
A2A3_A1A2 = double.Parse(line.Split(',')[index++]),

A1A2A3_Bypass = double.Parse(line.Split(',')[index++]),
A1A2A3_A1 = double.Parse(line.Split(',')[index++]),
A1A2A3_A2 = double.Parse(line.Split(',')[index++]),
A1A2A3_A1A2 = double.Parse(line.Split(',')[index++]),
});
}

if (info.CalVersion.ToUpper() == "V5") index = 32;
else index = 29;

if (!calPortCalData.Any(x => (x.Key.Frequency == srcFreq && x.Key.SigGen == srcSelect &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulationType && x.Key.Waveform ==
modulationFile && x.Key.Level == srcLevel)))
{
PortModuleUserCalSourceConfig calKey = new PortModuleUserCalSourceConfig() { Frequency =
srcFreq, SigGen = srcSelect, SrcPort = srcPort, Modulation = modulationType, Waveform =
modulationFile, Level = srcLevel };
calPortCalDataWithLevel.Add(calKey, new PortModuleUserCalDirectData()
{
Bypass = double.Parse(line.Split(',')[index++]),
Bypass_A1 = double.Parse(line.Split(',')[index++]),
Bypass_A2 = double.Parse(line.Split(',')[index++]),
Bypass_A1A2 = double.Parse(line.Split(',')[index++]),

A1_Bypass = double.Parse(line.Split(',')[index++]),
A1_A1 = double.Parse(line.Split(',')[index++]),
A1_A2 = double.Parse(line.Split(',')[index++]),
A1_A1A2 = double.Parse(line.Split(',')[index++]),

A2_Bypass = double.Parse(line.Split(',')[index++]),
A2_A1 = double.Parse(line.Split(',')[index++]),
A2_A2 = double.Parse(line.Split(',')[index++]),
A2_A1A2 = double.Parse(line.Split(',')[index++]),

A3_Bypass = double.Parse(line.Split(',')[index++]),
A3_A1 = double.Parse(line.Split(',')[index++]),
A3_A2 = double.Parse(line.Split(',')[index++]),
A3_A1A2 = double.Parse(line.Split(',')[index++]),
```

```
A1A2_Bypass = double.Parse(line.Split(',')[index++]),
A1A2_A1 = double.Parse(line.Split(',')[index++]),
A1A2_A2 = double.Parse(line.Split(',')[index++]),
A1A2_A1A2 = double.Parse(line.Split(',')[index++]),

A1A3_Bypass = double.Parse(line.Split(',')[index++]),
A1A3_A1 = double.Parse(line.Split(',')[index++]),
A1A3_A2 = double.Parse(line.Split(',')[index++]),
A1A3_A1A2 = double.Parse(line.Split(',')[index++]),

A2A3_Bypass = double.Parse(line.Split(',')[index++]),
A2A3_A1 = double.Parse(line.Split(',')[index++]),
A2A3_A2 = double.Parse(line.Split(',')[index++]),
A2A3_A1A2 = double.Parse(line.Split(',')[index++]),

A1A2A3_Bypass = double.Parse(line.Split(',')[index++]),
A1A2A3_A1 = double.Parse(line.Split(',')[index++]),
A1A2A3_A2 = double.Parse(line.Split(',')[index++]),
A1A2A3_A1A2 = double.Parse(line.Split(',')[index++]),
});
}
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
return -1;
}

// Added for dynamic waveform calibration support (User Calibration v5.0 and up)
//if (info.CalVersion.ToUpper() == "V5")
//{
//    CalImport import = new CalImport();
//    import.ReadWaveSettingsFile(ref waveData);

//    for (int i = 0; i < waveData.Count; i++)
//    {
//        modulationFile = waveData[i].WaveConfig.WaveformFile;
//        marker = waveData[i].WaveConfig.Marker;
```

```
//        if (!waveConfig.Any(x => (x.Key.WaveformFile == modulationFile && x.Key.Marker ==
marker)))
//        {
//            WaveformKey waveKey = new WaveformKey() { WaveformFile = modulationFile, Marker
= (short)marker };
//            waveConfig.Add(waveKey, new WaveformSettings()
//            {
//                WaveformFile = waveData[i].WaveConfig.WaveformFile,
//                Modulation = waveData[i].WaveConfig.Modulation,
//                Bandwidth = waveData[i].WaveConfig.Bandwidth,
//                Marker = waveData[i].WaveConfig.Marker,
//                MarkerLocation = waveData[i].WaveConfig.MarkerLocation,
//                DataLocation = waveData[i].WaveConfig.DataLocation,
//                DataLength = waveData[i].WaveConfig.DataLength,
//                MeasLength = waveData[i].WaveConfig.MeasLength,
//                MeasOffset = waveData[i].WaveConfig.MeasOffset,
//                DutyCycle = waveData[i].WaveConfig.DutyCycle,
//                FrameLength = waveData[i].WaveConfig.FrameLength,
//                MeasSpan = waveData[i].WaveConfig.MeasSpan,
//                MinSampleFreq = waveData[i].WaveConfig.MinSampleFreq,
//                RecSampleFreq = waveData[i].WaveConfig.RecSampleFreq,
//                AnalysisSlot = waveData[i].WaveConfig.AnalysisSlot,
//                Headroom = waveData[i].WaveConfig.Headroom,

//                // Waveform marker information
//                //MarkerData iqsMarkerData;
//                //iqsMarkerData = waveData[i].WaveConfig.iqsMarkerData.
//            });
//        }
//    }
//}

return 0;
}

public double GetSrcCalFactor(double freq, SigGen sigGen, SrcPort srcPort, SrcAmpConfig amp,
string modulation)
{
double calFactor = double.NaN;

PortModuleUserCalSourceData calData = GetSrcCalFactorForAmpConfig(freq, sigGen, srcPort,
```

```
modulation);

if (calData != null)
{
switch (amp)
{
case SrcAmpConfig.Bypass:
calFactor = calData.Bypass;
break;

case SrcAmpConfig.A1:
calFactor = calData.A1;
break;

case SrcAmpConfig.A2:
calFactor = calData.A2;
break;

case SrcAmpConfig.A3:
calFactor = calData.A3;
break;

case SrcAmpConfig.A1_A2:
calFactor = calData.A1_A2;
break;

case SrcAmpConfig.A1_A3:
calFactor = calData.A1_A3;
break;

case SrcAmpConfig.A2_A3:
calFactor = calData.A2_A3;
break;

case SrcAmpConfig.A1_A2_A3:
calFactor = calData.A1_A2_A3;
break;

case SrcAmpConfig.None:
calFactor = calData.Bypass;
break;
}
```

```
}

return calFactor;
}

public double GetSrcCalFactor(double freq, SigGen sigGen, SrcPort srcPort, SrcAmpConfig amp,
string modulation, double level)
{
double calFactor = double.NaN;

PortModuleUserCalSourceData calData = GetSrcCalFactorForAmpConfig(freq, sigGen, srcPort,
modulation, level);

if (calData != null)
{
switch (amp)
{
case SrcAmpConfig.Bypass:
calFactor = calData.Bypass;
break;

case SrcAmpConfig.A1:
calFactor = calData.A1;
break;

case SrcAmpConfig.A2:
calFactor = calData.A2;
break;

case SrcAmpConfig.A3:
calFactor = calData.A3;
break;

case SrcAmpConfig.A1_A2:
calFactor = calData.A1_A2;
break;

case SrcAmpConfig.A1_A3:
calFactor = calData.A1_A3;
break;

case SrcAmpConfig.A2_A3:
```

```
calFactor = calData.A2_A3;
break;

case SrcAmpConfig.A1_A2_A3:
calFactor = calData.A1_A2_A3;
break;

case SrcAmpConfig.None:
calFactor = calData.Bypass;
break;
}
}

return calFactor;
}

public double GetSrcCalFactor(double freq, SigGen sigGen, SrcPort srcPort, SrcAmpConfig amp,
string modulation, string waveform)
{
double calFactor = double.NaN;
waveform = Path.GetFileNameWithoutExtension(waveform) + ".aiq";
PortModuleUserCalSourceData calData = GetSrcCalFactorForAmpConfig(freq, sigGen, srcPort,
modulation, waveform);

if (calData != null)
{
switch (amp)
{
case SrcAmpConfig.Bypass:
calFactor = calData.Bypass;
break;

case SrcAmpConfig.A1:
calFactor = calData.A1;
break;

case SrcAmpConfig.A2:
calFactor = calData.A2;
break;

case SrcAmpConfig.A3:
calFactor = calData.A3;
```

```
break;

case SrcAmpConfig.A1_A2:
calFactor = calData.A1_A2;
break;

case SrcAmpConfig.A1_A3:
calFactor = calData.A1_A3;
break;

case SrcAmpConfig.A2_A3:
calFactor = calData.A2_A3;
break;

case SrcAmpConfig.A1_A2_A3:
calFactor = calData.A1_A2_A3;
break;

case SrcAmpConfig.None:
calFactor = calData.Bypass;
break;
}
}

return calFactor;
}

public double GetSrcCalFactor(double freq, SigGen sigGen, SrcPort srcPort, SrcAmpConfig amp,
string modulation, string waveform, double level)
{
double calFactor = double.NaN;
waveform = Path.GetFileNameWithoutExtension(waveform) + ".aiq";
PortModuleUserCalSourceData calData = GetSrcCalFactorForAmpConfig(freq, sigGen, srcPort,
modulation, waveform, level);

if (calData != null)
{
switch (amp)
{
case SrcAmpConfig.Bypass:
calFactor = calData.Bypass;
break;
```

```
case SrcAmpConfig.A1:
calFactor = calData.A1;
break;

case SrcAmpConfig.A2:
calFactor = calData.A2;
break;

case SrcAmpConfig.A3:
calFactor = calData.A3;
break;

case SrcAmpConfig.A1_A2:
calFactor = calData.A1_A2;
break;

case SrcAmpConfig.A1_A3:
calFactor = calData.A1_A3;
break;

case SrcAmpConfig.A2_A3:
calFactor = calData.A2_A3;
break;

case SrcAmpConfig.A1_A2_A3:
calFactor = calData.A1_A2_A3;
break;

case SrcAmpConfig.None:
calFactor = calData.Bypass;
break;
}
}

return calFactor;
}

public double GetMeasCalFactor(double freq, MeasPort measPort, MeasFilt filter,
SrcMeasAmpConfig amp, string modulation)
{
double calFactor = double.NaN;
```

```
PortModuleUserCalMeasureData calData = GetMeasCalFactorForAmpConfig(freq, measPort,
filter, modulation);

if (calData != null)
{
switch (amp)
{
case SrcMeasAmpConfig.Bypass:
calFactor = calData.Bypass;
break;

case SrcMeasAmpConfig.A1:
calFactor = calData.A1;
break;

case SrcMeasAmpConfig.A2:
calFactor = calData.A2;
break;

case SrcMeasAmpConfig.A1_A2:
calFactor = calData.A1_A2;
break;

case SrcMeasAmpConfig.A4_MEAS:
calFactor = calData.A4_MEAS;
break;

case SrcMeasAmpConfig.A4_MEAS_A1:
calFactor = calData.A4_MEAS_A1;
break;

case SrcMeasAmpConfig.A4_MEAS_A2:
calFactor = calData.A4_MEAS_A2;
break;

case SrcMeasAmpConfig.A4_MEAS_A1_A2:
calFactor = calData.A4_MEAS_A1_A2;
break;
}
}
```

```
return calFactor;
}

public double GetMeasCalFactor(double freq, MeasPort measPort, MeasFilt filter,
SrcMeasAmpConfig amp, string modulation, double level)
{
double calFactor = double.NaN;

PortModuleUserCalMeasureData calData = GetMeasCalFactorForAmpConfig(freq, measPort,
filter, modulation, level);

if (calData != null)
{
switch (amp)
{
case SrcMeasAmpConfig.Bypass:
calFactor = calData.Bypass;
break;

case SrcMeasAmpConfig.A1:
calFactor = calData.A1;
break;

case SrcMeasAmpConfig.A2:
calFactor = calData.A2;
break;

case SrcMeasAmpConfig.A1_A2:
calFactor = calData.A1_A2;
break;

case SrcMeasAmpConfig.A4_MEAS:
calFactor = calData.A4_MEAS;
break;

case SrcMeasAmpConfig.A4_MEAS_A1:
calFactor = calData.A4_MEAS_A1;
break;

case SrcMeasAmpConfig.A4_MEAS_A2:
calFactor = calData.A4_MEAS_A2;
break;
```

```
case SrcMeasAmpConfig.A4_MEAS_A1_A2:
calFactor = calData.A4_MEAS_A1_A2;
break;
}
}

return calFactor;
}

public double GetMeasCalFactor(double freq, MeasPort measPort, MeasFilt filter,
SrcMeasAmpConfig amp, string modulation, string waveform)
{
double calFactor = double.NaN;
waveform = Path.GetFileNameWithoutExtension(waveform) + ".aiq";
PortModuleUserCalMeasureData calData = GetMeasCalFactorForAmpConfig(freq, measPort,
filter, modulation, waveform);

if (calData != null)
{
switch (amp)
{
case SrcMeasAmpConfig.Bypass:
calFactor = calData.Bypass;
break;

case SrcMeasAmpConfig.A1:
calFactor = calData.A1;
break;

case SrcMeasAmpConfig.A2:
calFactor = calData.A2;
break;

case SrcMeasAmpConfig.A1_A2:
calFactor = calData.A1_A2;
break;

case SrcMeasAmpConfig.A4_MEAS:
calFactor = calData.A4_MEAS;
break;
```

```
case SrcMeasAmpConfig.A4_MEAS_A1:
calFactor = calData.A4_MEAS_A1;
break;

case SrcMeasAmpConfig.A4_MEAS_A2:
calFactor = calData.A4_MEAS_A2;
break;

case SrcMeasAmpConfig.A4_MEAS_A1_A2:
calFactor = calData.A4_MEAS_A1_A2;
break;
}
}

return calFactor;
}

public double GetMeasCalFactor(double freq, MeasPort measPort, MeasFilt filter,
SrcMeasAmpConfig amp, string modulation, string waveform, double level)
{
double calFactor = double.NaN;
waveform = Path.GetFileNameWithoutExtension(waveform) + ".aiq";
PortModuleUserCalMeasureData calData = GetMeasCalFactorForAmpConfig(freq, measPort,
filter, modulation, waveform, level);

if (calData != null)
{
switch (amp)
{
case SrcMeasAmpConfig.Bypass:
calFactor = calData.Bypass;
break;

case SrcMeasAmpConfig.A1:
calFactor = calData.A1;
break;

case SrcMeasAmpConfig.A2:
calFactor = calData.A2;
break;

case SrcMeasAmpConfig.A1_A2:
```

```
calFactor = calData.A1_A2;
break;

case SrcMeasAmpConfig.A4_MEAS:
calFactor = calData.A4_MEAS;
break;

case SrcMeasAmpConfig.A4_MEAS_A1:
calFactor = calData.A4_MEAS_A1;
break;

case SrcMeasAmpConfig.A4_MEAS_A2:
calFactor = calData.A4_MEAS_A2;
break;

case SrcMeasAmpConfig.A4_MEAS_A1_A2:
calFactor = calData.A4_MEAS_A1_A2;
break;
}
}

return calFactor;
}

public double GetDirCalFactor(double freq, SigGen sigGen, SrcPort srcPort, SrcAmpConfig amp1,
MeasAmpConfig amp2, string modulation)
{
double calFactor = double.NaN;

PortModuleUserCalDirectData calData = GetDirectCalFactorForAmpConfig(freq, sigGen, srcPort,
modulation);

if (calData != null)
{
if (amp1 == SrcAmpConfig.Bypass)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.Bypass_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.Bypass_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.Bypass_A1A2;
}
else if (amp1 == SrcAmpConfig.A1)
```

```
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1_A1A2;
}
else if (amp1 == SrcAmpConfig.A2)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A2_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A2_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A2_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A2_A1A2;
}
else if (amp1 == SrcAmpConfig.A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A2)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A2_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A2_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A2_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A2_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A2_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A2A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A2A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A2A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A2A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A2_A3)
```

```
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A2A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A2A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A2A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A2A3_A1A2;
}
}

return calFactor;
}

public double GetDirCalFactor(double freq, SigGen sigGen, SrcPort srcPort, SrcAmpConfig amp1,
MeasAmpConfig amp2, string modulation, double level)
{
double calFactor = double.NaN;

PortModuleUserCalDirectData calData = GetDirectCalFactorForAmpConfig(freq, sigGen, srcPort,
modulation, level);

if (calData != null)
{
if (amp1 == SrcAmpConfig.Bypass)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.Bypass_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.Bypass_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.Bypass_A1A2;
}
else if (amp1 == SrcAmpConfig.A1)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1_A1A2;
}
else if (amp1 == SrcAmpConfig.A2)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A2_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A2_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A2_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A2_A1A2;
}
```

```
else if (amp1 == SrcAmpConfig.A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A2)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A2_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A2_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A2_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A2_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A2_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A2A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A2A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A2A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A2A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A2_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A2A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A2A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A2A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A2A3_A1A2;
}
}

return calFactor;
}
```

public double GetDirCalFactor(double freq, SigGen sigGen, SrcPort srcPort, SrcAmpConfig amp1, MeasAmpConfig amp2, string modulation, string waveform)

```
{
double calFactor = double.NaN;
waveform = Path.GetFileNameWithoutExtension(waveform) + ".aiq";
PortModuleUserCalDirectData calData = GetDirectCalFactorForAmpConfig(freq, sigGen, srcPort,
modulation, waveform);

if (calData != null)
{
if (amp1 == SrcAmpConfig.Bypass)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.Bypass_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.Bypass_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.Bypass_A1A2;
}
else if (amp1 == SrcAmpConfig.A1)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1_A1A2;
}
else if (amp1 == SrcAmpConfig.A2)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A2_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A2_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A2_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A2_A1A2;
}
else if (amp1 == SrcAmpConfig.A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A2)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A2_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A2_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A2_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A2_A1A2;
```

```
}
else if (amp1 == SrcAmpConfig.A1_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A2_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A2A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A2A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A2A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A2A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A2_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A2A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A2A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A2A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A2A3_A1A2;
}
}

return calFactor;
}

public double GetDirCalFactor(double freq, SigGen sigGen, SrcPort srcPort, SrcAmpConfig amp1,
MeasAmpConfig amp2, string modulation, string waveform, double level)
{
double calFactor = double.NaN;
waveform = Path.GetFileNameWithoutExtension(waveform) + ".aiq";
PortModuleUserCalDirectData calData = GetDirectCalFactorForAmpConfig(freq, sigGen, srcPort,
modulation, waveform, level);

if (calData != null)
{
if (amp1 == SrcAmpConfig.Bypass)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.Bypass_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.Bypass_A2;
```

```
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.Bypass_A1A2;
}
else if (amp1 == SrcAmpConfig.A1)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1_A1A2;
}
else if (amp1 == SrcAmpConfig.A2)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A2_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A2_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A2_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A2_A1A2;
}
else if (amp1 == SrcAmpConfig.A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A2)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A2_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A2_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A2_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A2_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A2_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A2A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A2A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A2A3_A2;
```

```
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A2A3_A1A2;
}
else if (amp1 == SrcAmpConfig.A1_A2_A3)
{
if (amp2 == MeasAmpConfig.Bypass) calFactor = calData.A1A2A3_Bypass;
else if (amp2 == MeasAmpConfig.A1) calFactor = calData.A1A2A3_A1;
else if (amp2 == MeasAmpConfig.A2) calFactor = calData.A1A2A3_A2;
else if (amp2 == MeasAmpConfig.A1_A2) calFactor = calData.A1A2A3_A1A2;
}
}

return calFactor;
}

public PortModuleUserCalSourceData GetSrcCalFactorForAmpConfig(double freq, SigGen
sigGen, SrcPort srcPort, string modulation)
{
if (sourcePortCalData.Any(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulation)))
{
return sourcePortCalData.First(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulation)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

public PortModuleUserCalSourceData GetSrcCalFactorForAmpConfig(double freq, SigGen
sigGen, SrcPort srcPort, string modulation, double level)
{
if (sourcePortCalDataWithLevel.Any(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen
&& x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Level == level)))
{
return sourcePortCalDataWithLevel.First(x => (x.Key.Frequency == freq && x.Key.SigGen ==
sigGen && x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Level ==
level)).Value;
}
else
{
```

```
//todo: add diagnostic logging
return null;
}
}

public PortModuleUserCalSourceData GetSrcCalFactorForAmpConfig(double freq, SigGen
sigGen, SrcPort srcPort, string modulation, string waveform)
{
if (sourcePortCalDataWithLevel.Any(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen
&& x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Waveform ==
waveform)))
{
return sourcePortCalDataWithLevel.First(x => (x.Key.Frequency == freq && x.Key.SigGen ==
sigGen && x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Waveform ==
waveform)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

public PortModuleUserCalSourceData GetSrcCalFactorForAmpConfig(double freq, SigGen
sigGen, SrcPort srcPort, string modulation, string waveform, double level)
{
if (sourcePortCalDataWithLevel.Any(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen
&& x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Waveform ==
waveform && x.Key.Level == level)))
{
return sourcePortCalDataWithLevel.First(x => (x.Key.Frequency == freq && x.Key.SigGen ==
sigGen && x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Waveform ==
waveform && x.Key.Level == level)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

public PortModuleUserCalMeasureData GetMeasCalFactorForAmpConfig(double freq, MeasPort
```

```
measPort, MeasFilt filter, string modulation)
{
if (measurePortCalData.Any(x => (x.Key.Frequency == freq && x.Key.MeasPort == measPort &&
x.Key.Attenuator == filter && x.Key.Modulation == modulation)))
{
return measurePortCalData.First(x => (x.Key.Frequency == freq && x.Key.MeasPort == measPort
&& x.Key.Attenuator == filter && x.Key.Modulation == modulation)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

public PortModuleUserCalMeasureData GetMeasCalFactorForAmpConfig(double freq, MeasPort
measPort, MeasFilt filter, string modulation, double level)
{
if (measurePortCalDataWithLevel.Any(x => (x.Key.Frequency == freq && x.Key.MeasPort ==
measPort && x.Key.Attenuator == filter && x.Key.Modulation == modulation && x.Key.Level ==
level)))
{
return measurePortCalDataWithLevel.First(x => (x.Key.Frequency == freq && x.Key.MeasPort ==
measPort && x.Key.Attenuator == filter && x.Key.Modulation == modulation && x.Key.Level ==
level)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

public PortModuleUserCalMeasureData GetMeasCalFactorForAmpConfig(double freq, MeasPort
measPort, MeasFilt filter, string modulation, string waveform)
{
if (measurePortCalData.Any(x => (x.Key.Frequency == freq && x.Key.MeasPort == measPort &&
x.Key.Attenuator == filter && x.Key.Modulation == modulation && x.Key.Waveform == waveform)))
{
return measurePortCalData.First(x => (x.Key.Frequency == freq && x.Key.MeasPort == measPort
&& x.Key.Attenuator == filter && x.Key.Modulation == modulation && x.Key.Waveform ==
waveform)).Value;
```

```
}
else
{
//todo: add diagnostic logging
return null;
}
}

public PortModuleUserCalMeasureData GetMeasCalFactorForAmpConfig(double freq, MeasPort measPort, MeasFilt filter, string modulation, string waveform, double level)
{
if (measurePortCalDataWithLevel.Any(x => (x.Key.Frequency == freq && x.Key.MeasPort == measPort && x.Key.Attenuator == filter && x.Key.Modulation == modulation && x.Key.Waveform == waveform && x.Key.Level == level)))
{
return measurePortCalDataWithLevel.First(x => (x.Key.Frequency == freq && x.Key.MeasPort == measPort && x.Key.Attenuator == filter && x.Key.Modulation == modulation && x.Key.Waveform == waveform && x.Key.Level == level)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

public PortModuleUserCalDirectData GetDirectCalFactorForAmpConfig(double freq, SigGen sigGen, SrcPort srcPort, string modulation)
{
if (calPortCalData.Any(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen && x.Key.SrcPort == srcPort && x.Key.Modulation == modulation)))
{
return calPortCalData.First(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen && x.Key.SrcPort == srcPort && x.Key.Modulation == modulation)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}
```

```
public PortModuleUserCalDirectData GetDirectCalFactorForAmpConfig(double freq, SigGen
sigGen, SrcPort srcPort, string modulation, double level)
{
if (calPortCalDataWithLevel.Any(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Level == level)))
{
return calPortCalDataWithLevel.First(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen
&& x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Level == level)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

public PortModuleUserCalDirectData GetDirectCalFactorForAmpConfig(double freq, SigGen
sigGen, SrcPort srcPort, string modulation, string waveform)
{
if (calPortCalData.Any(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Waveform == waveform)))
{
return calPortCalData.First(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Waveform ==
waveform)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

public PortModuleUserCalDirectData GetDirectCalFactorForAmpConfig(double freq, SigGen
sigGen, SrcPort srcPort, string modulation, string waveform, double level)
{
if (calPortCalDataWithLevel.Any(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen &&
x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Waveform == waveform
&& x.Key.Level == level)))
{
return calPortCalDataWithLevel.First(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen
&& x.Key.SrcPort == srcPort && x.Key.Modulation == modulation && x.Key.Waveform ==
```

```
waveform && x.Key.Level == level)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}
}


//========================= Switch Filter / HPA Calibration
=========================

public class CalDataRF300
{
public enum AmpPort { HPA1, HPA2, HPA3, HPA4, None };
public enum sfCouplerPort { DUT_P1_FWD, DUT_P2_FWD, DUT_P3_FWD, DUT_P4_FWD,
DUT_P5_FWD, DUT_P6_FWD, DUT_P7_FWD, DUT_P8_FWD, DUT_P1_REV, DUT_P2_REV,
DUT_P3_REV, DUT_P4_REV, DUT_P5_REV, DUT_P6_REV, DUT_P7_REV, DUT_P8_REV,
None };
public enum sfPort { DUT_P1, DUT_P1_FWD, DUT_P1_REV, DUT_P2, DUT_P2_FWD,
DUT_P2_REV, DUT_P3, DUT_P3_FWD, DUT_P3_REV, DUT_P4, DUT_P4_FWD,
DUT_P4_REV, DUT_P5, DUT_P5_FWD, DUT_P5_REV, DUT_P6, DUT_P6_FWD,
DUT_P6_REV, DUT_P7, DUT_P7_FWD, DUT_P7_REV, DUT_P8, DUT_P8_FWD,
DUT_P8_REV, DUT_P1_P4_COUPLED, DUT_P5_P8_COUPLED, BB1, BB2, BB3, BB4,
HB1_HPF, HB1_LPF, LB1_HPF, LB1_LPF, HB2_HPF, HB2_LPF, LB2_HPF, LB2_LPF, DM1,
DM2, DM3, DM4, None }

public int record = 0;
public bool copy = false;
public bool measCalPath = false;
public SigGen srcSelect;
public SrcPort srcPort;
public AmpPort ampPort;
public sfPort srcSwitchFilterPrimary;
public sfPort srcSwitchFilterSecondary;
public sfCouplerPort srcCouplerPort;
public sfPort measSwitchFilterPrimary;
public sfPort measSwitchFilterSecondary;
public sfCouplerPort measCouplerPort;
public MeasPort measPort;
```

```
public MeasFilt measFilter;
public string comment = "";
public string srcPortAlias = "";
public string measPortAlias = "";
public string modulationType = "";
public string modulationFile = "";
public double srcFreq = 0;
public double srcLevel = -100;
public double dutyCycle = 100;
public double measAtten = 0;
public double[] srcCalFactor = new double[8];
public double[] measCalFactor = new double[4];
public double[] calCalFactor = new double[32];
public double[] srcCouplerCalFactor = new double[2];
public double[] measCouplerCalFactor = new double[2];
public double noiseCalFactorSrc = 0;
public double noiseCalFactorSrcPath = 0;
public double noiseCalFactorOn = 0;
public double noiseCalFactorOff = 0;
public double noiseCalFactorLoss = 0;
public double noiseCalFactorENR = 0;
}

public class CoupledPort
{
public MeasPort DUT_P1_P4;
public MeasPort DUT_P5_P8;
}

public class SwitchSelect
{
public enum SwitchPosition { BB, FILT, HB, HB_Deselect, LB_HPF, None }

public SwitchPosition[] switchPosition = new SwitchPosition[8];
}

public class SwitchFilterUserCalSourceConfig
{
public double Frequency { get; set; }
public SigGen SigGen { get; set; }
public SrcPort SrcPort { get; set; }
public CalDataRF300.AmpPort AmpPort { get; set; }
```

```
public CalDataRF300.sfPort SwitchFilterPrimary { get; set; }
public CalDataRF300.sfPort SwitchFilterSecondary { get; set; }
public CalDataRF300.sfCouplerPort SwitchFilterCoupler { get; set; }
}

public class SwitchFilterUserCalSourceData
{
public double A1 { get; set; }
public double A2 { get; set; }
public double A3 { get; set; }
public double A1_A2 { get; set; }
public double A1_A3 { get; set; }
public double A2_A3 { get; set; }
public double A1_A2_A3 { get; set; }
public double Bypass { get; set; }
public double SF_Amp_Off { get; set; }
public double SF_Amp_On { get; set; }

public double CAL_Bypass { get; set; }
public double CAL_A1 { get; set; }
public double CAL_A2 { get; set; }
public double CAL_A3 { get; set; }
public double CAL_A1_A2 { get; set; }
public double CAL_A1_A3 { get; set; }
public double CAL_A2_A3 { get; set; }
public double CAL_A1_A2_A3 { get; set; }
}

public class SwitchFilterUserCalMeasureConfig
{
public double Frequency { get; set; }
public MeasPort MeasPort { get; set; }
public CalDataRF300.sfPort SwitchFilterPrimary { get; set; }
public CalDataRF300.sfPort SwitchFilterSecondary { get; set; }
public CalDataRF300.sfCouplerPort SwitchFilterCoupler { get; set; }
public MeasFilt MeasFilt { get; set; }
}

public class SwitchFilterUserCalMeasureData
{
public double A1 { get; set; }
public double A2 { get; set; }
```

```
public double A1_A2 { get; set; }
public double Bypass { get; set; }
public double SF_Amp_Off { get; set; }
public double SF_Amp_On { get; set; }
}

public class SwitchFilterSystemCal
{
private bool writeToConsole = true;

public Info info = new Info();
public AmpData ampData = new AmpData();
public CoupledPort coupledPort = new CoupledPort();
public SwitchSelect switchSelect = new SwitchSelect();
public PortModuleAttenuation atten = new PortModuleAttenuation();

public Dictionary<SwitchFilterUserCalSourceConfig, SwitchFilterUserCalSourceData>
sourcePortCalData = new Dictionary<SwitchFilterUserCalSourceConfig,
SwitchFilterUserCalSourceData>();
public Dictionary<SwitchFilterUserCalMeasureConfig, SwitchFilterUserCalMeasureData>
measurePortCalData = new Dictionary<SwitchFilterUserCalMeasureConfig,
SwitchFilterUserCalMeasureData>();

public int ReadCalDataFile()
{
const string calDataFile =
@"C:\MerlinTest\MT_Home\Smithers\APS200_VC1899_RF_Cal_Data.csv";

if (writeToConsole) Console.WriteLine("Loading source port module calibration data file: {0}\n",
calDataFile);

string message = "";

// Check if file exists
if (!File.Exists(calDataFile))
{
message = "ERROR: Port module calibration data file not found:\n" + calDataFile;
if (writeToConsole) Console.WriteLine(message);
MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}
```

```
// Check if file is locked
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
}
catch (IOException ex)
{
MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Read cal data from file
ImportCalData(calDataFile);

return 0;
}

public int ReadCalDataFile(string calDataFile)
{
if (writeToConsole) Console.WriteLine("Loading source port module calibration data file: {0}\n",
calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "ERROR: Port module calibration data file not found:\n" + calDataFile;
if (writeToConsole) Console.WriteLine(message);
MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Check if file is locked
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
```

```
}
catch (IOException ex)
{
MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Read cal data from file
ImportCalData(calDataFile);

return 0;
}

public int ImportCalData(string calDataFile)
{
string line = "";
double freq = 0;
double level = 0;
double measAtten = 0;
string srcPortAlias = "";
string measPortAlias = "";
SigGen sigGen = SigGen.ISOLATE;
SrcPort srcPort = SrcPort.NONE;
MeasPort measPort = MeasPort.NONE;
MeasFilt filter = MeasFilt.NONE;
CalDataRF300.AmpPort ampPort = CalDataRF300.AmpPort.None;
CalDataRF300.sfPort srcPrimary = CalDataRF300.sfPort.None;
CalDataRF300.sfPort srcSecondary = CalDataRF300.sfPort.None;
CalDataRF300.sfCouplerPort srcCoupler = CalDataRF300.sfCouplerPort.None;
CalDataRF300.sfPort measPrimary = CalDataRF300.sfPort.None;
CalDataRF300.sfPort measSecondary = CalDataRF300.sfPort.None;
CalDataRF300.sfCouplerPort measCoupler = CalDataRF300.sfCouplerPort.None;

try
{
using (StreamReader stream = new StreamReader(calDataFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];
```

```
if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "REVISION:")
{
info.Revision = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "TEST PROGRAM:")
{
info.TesterId = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "COMMENT:")
{
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "DATE:")
{
info.Date = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "TESTER ID:")
{
info.TesterId = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "AMPLIFIER BIAS CURRENT:")
{
ampData.srcPortAmpBias[0] = double.Parse(line.Split(',')[1]);
ampData.srcPortAmpBias[1] = double.Parse(line.Split(',')[2]);
ampData.srcPortAmpBias[2] = double.Parse(line.Split(',')[3]);
ampData.srcPortAmpBias[3] = double.Parse(line.Split(',')[4]);
ampData.srcPortAmpBias[4] = double.Parse(line.Split(',')[5]);

ampData.measPortAmpBias[0] = double.Parse(line.Split(',')[6]);
ampData.measPortAmpBias[1] = double.Parse(line.Split(',')[7]);
}
else if (tmpString.ToUpper() == "AMPLIFIER BIAS TEMPERATURE:")
{
ampData.srcPortAmpTemp[0] = double.Parse(line.Split(',')[1]);
```

```
ampData.srcPortAmpTemp[1] = double.Parse(line.Split(',')[2]);
ampData.srcPortAmpTemp[2] = double.Parse(line.Split(',')[3]);
ampData.srcPortAmpTemp[3] = double.Parse(line.Split(',')[4]);
ampData.srcPortAmpTemp[4] = double.Parse(line.Split(',')[5]);

ampData.measPortAmpTemp[0] = double.Parse(line.Split(',')[6]);
ampData.measPortAmpTemp[1] = double.Parse(line.Split(',')[7]);
}
else if (tmpString.ToUpper() == "DUT_P1-P4 PORT:")
{
string port = line.Split(',')[1];

if (port.ToUpper() == "P1") coupledPort.DUT_P1_P4 = MeasPort.P1;
else if (port.ToUpper() == "P2") coupledPort.DUT_P1_P4 = MeasPort.P2;
else if (port.ToUpper() == "P3") coupledPort.DUT_P1_P4 = MeasPort.P3;
else if (port.ToUpper() == "P4") coupledPort.DUT_P1_P4 = MeasPort.P4;
else if (port.ToUpper() == "P5") coupledPort.DUT_P1_P4 = MeasPort.P5;
else if (port.ToUpper() == "P6") coupledPort.DUT_P1_P4 = MeasPort.P6;
else if (port.ToUpper() == "P7") coupledPort.DUT_P1_P4 = MeasPort.P7;
else if (port.ToUpper() == "P8") coupledPort.DUT_P1_P4 = MeasPort.P8;
else if (port.ToUpper() == "P9") coupledPort.DUT_P1_P4 = MeasPort.P9;
else if (port.ToUpper() == "P10") coupledPort.DUT_P1_P4 = MeasPort.P10;
else if (port.ToUpper() == "P11") coupledPort.DUT_P1_P4 = MeasPort.P11;
else if (port.ToUpper() == "P12") coupledPort.DUT_P1_P4 = MeasPort.P12;
else if (port.ToUpper() == "P13") coupledPort.DUT_P1_P4 = MeasPort.P13;
else if (port.ToUpper() == "P14") coupledPort.DUT_P1_P4 = MeasPort.P14;
else if (port.ToUpper() == "P15") coupledPort.DUT_P1_P4 = MeasPort.P15;
else if (port.ToUpper() == "P16") coupledPort.DUT_P1_P4 = MeasPort.P16;

else if (port.ToUpper() == "P1_FWD") coupledPort.DUT_P1_P4 = MeasPort.P1_Fwd;
else if (port.ToUpper() == "P2_FWD") coupledPort.DUT_P1_P4 = MeasPort.P2_Fwd;
else if (port.ToUpper() == "P3_FWD") coupledPort.DUT_P1_P4 = MeasPort.P3_Fwd;
else if (port.ToUpper() == "P4_FWD") coupledPort.DUT_P1_P4 = MeasPort.P4_Fwd;
else if (port.ToUpper() == "P5_FWD") coupledPort.DUT_P1_P4 = MeasPort.P5_Fwd;
else if (port.ToUpper() == "P6_FWD") coupledPort.DUT_P1_P4 = MeasPort.P6_Fwd;
else if (port.ToUpper() == "P7_FWD") coupledPort.DUT_P1_P4 = MeasPort.P7_Fwd;
else if (port.ToUpper() == "P8_FWD") coupledPort.DUT_P1_P4 = MeasPort.P8_Fwd;

else if (port.ToUpper() == "P1_REV") coupledPort.DUT_P1_P4 = MeasPort.P1_Rev;
else if (port.ToUpper() == "P2_REV") coupledPort.DUT_P1_P4 = MeasPort.P2_Rev;
else if (port.ToUpper() == "P3_REV") coupledPort.DUT_P1_P4 = MeasPort.P3_Rev;
else if (port.ToUpper() == "P4_REV") coupledPort.DUT_P1_P4 = MeasPort.P4_Rev;
```

```
else if (port.ToUpper() == "P5_REV") coupledPort.DUT_P1_P4 = MeasPort.P5_Rev;
else if (port.ToUpper() == "P6_REV") coupledPort.DUT_P1_P4 = MeasPort.P6_Rev;
else if (port.ToUpper() == "P7_REV") coupledPort.DUT_P1_P4 = MeasPort.P7_Rev;
else if (port.ToUpper() == "P8_REV") coupledPort.DUT_P1_P4 = MeasPort.P8_Rev;

else if (port.ToUpper() == "M1") coupledPort.DUT_P1_P4 = MeasPort.M1;
else if (port.ToUpper() == "M2") coupledPort.DUT_P1_P4 = MeasPort.M2;
else if (port.ToUpper() == "M3") coupledPort.DUT_P1_P4 = MeasPort.M3;
else if (port.ToUpper() == "M4") coupledPort.DUT_P1_P4 = MeasPort.M4;
else if (port.ToUpper() == "M5") coupledPort.DUT_P1_P4 = MeasPort.M5;
else if (port.ToUpper() == "M6") coupledPort.DUT_P1_P4 = MeasPort.M6;
else if (port.ToUpper() == "M7") coupledPort.DUT_P1_P4 = MeasPort.M7;
else if (port.ToUpper() == "M8") coupledPort.DUT_P1_P4 = MeasPort.M8;
else if (port.ToUpper() == "M9") coupledPort.DUT_P1_P4 = MeasPort.M9;
else if (port.ToUpper() == "M10") coupledPort.DUT_P1_P4 = MeasPort.M10;
else if (port.ToUpper() == "M11") coupledPort.DUT_P1_P4 = MeasPort.M11;
else if (port.ToUpper() == "M12") coupledPort.DUT_P1_P4 = MeasPort.M12;
else if (port.ToUpper() == "M13") coupledPort.DUT_P1_P4 = MeasPort.M13;
else if (port.ToUpper() == "M14") coupledPort.DUT_P1_P4 = MeasPort.M14;
else if (port.ToUpper() == "M15") coupledPort.DUT_P1_P4 = MeasPort.M15;
else if (port.ToUpper() == "M16") coupledPort.DUT_P1_P4 = MeasPort.M16;
else if (port.ToUpper() == "M17") coupledPort.DUT_P1_P4 = MeasPort.M17;
else if (port.ToUpper() == "M18") coupledPort.DUT_P1_P4 = MeasPort.M18;
else if (port.ToUpper() == "M19") coupledPort.DUT_P1_P4 = MeasPort.M19;
else if (port.ToUpper() == "M20") coupledPort.DUT_P1_P4 = MeasPort.M20;
else if (port.ToUpper() == "M21") coupledPort.DUT_P1_P4 = MeasPort.M21;
else if (port.ToUpper() == "M22") coupledPort.DUT_P1_P4 = MeasPort.M22;
else if (port.ToUpper() == "M23") coupledPort.DUT_P1_P4 = MeasPort.M23;
else if (port.ToUpper() == "M24") coupledPort.DUT_P1_P4 = MeasPort.M24;
else if (port.ToUpper() == "M25") coupledPort.DUT_P1_P4 = MeasPort.M25;
else if (port.ToUpper() == "M26") coupledPort.DUT_P1_P4 = MeasPort.M26;

else if (port.ToUpper() == "MA") coupledPort.DUT_P1_P4 = MeasPort.MA;
else if (port.ToUpper() == "MB") coupledPort.DUT_P1_P4 = MeasPort.MB;
else if (port.ToUpper() == "MC") coupledPort.DUT_P1_P4 = MeasPort.MC;
else if (port.ToUpper() == "MD") coupledPort.DUT_P1_P4 = MeasPort.MD;

else coupledPort.DUT_P1_P4 = MeasPort.NONE;
}
else if (tmpString.ToUpper() == "DUT_P5-P8 PORT:")
{
string port = line.Split(',')[1];
```

```
if (port.ToUpper() == "P1") coupledPort.DUT_P5_P8 = MeasPort.P1;
else if (port.ToUpper() == "P2") coupledPort.DUT_P5_P8 = MeasPort.P2;
else if (port.ToUpper() == "P3") coupledPort.DUT_P5_P8 = MeasPort.P3;
else if (port.ToUpper() == "P4") coupledPort.DUT_P5_P8 = MeasPort.P4;
else if (port.ToUpper() == "P5") coupledPort.DUT_P5_P8 = MeasPort.P5;
else if (port.ToUpper() == "P6") coupledPort.DUT_P5_P8 = MeasPort.P6;
else if (port.ToUpper() == "P7") coupledPort.DUT_P5_P8 = MeasPort.P7;
else if (port.ToUpper() == "P8") coupledPort.DUT_P5_P8 = MeasPort.P8;
else if (port.ToUpper() == "P9") coupledPort.DUT_P5_P8 = MeasPort.P9;
else if (port.ToUpper() == "P10") coupledPort.DUT_P5_P8 = MeasPort.P10;
else if (port.ToUpper() == "P11") coupledPort.DUT_P5_P8 = MeasPort.P11;
else if (port.ToUpper() == "P12") coupledPort.DUT_P5_P8 = MeasPort.P12;
else if (port.ToUpper() == "P13") coupledPort.DUT_P5_P8 = MeasPort.P13;
else if (port.ToUpper() == "P14") coupledPort.DUT_P5_P8 = MeasPort.P14;
else if (port.ToUpper() == "P15") coupledPort.DUT_P5_P8 = MeasPort.P15;
else if (port.ToUpper() == "P16") coupledPort.DUT_P5_P8 = MeasPort.P16;

else if (port.ToUpper() == "P1_FWD") coupledPort.DUT_P5_P8 = MeasPort.P1_Fwd;
else if (port.ToUpper() == "P2_FWD") coupledPort.DUT_P5_P8 = MeasPort.P2_Fwd;
else if (port.ToUpper() == "P3_FWD") coupledPort.DUT_P5_P8 = MeasPort.P3_Fwd;
else if (port.ToUpper() == "P4_FWD") coupledPort.DUT_P5_P8 = MeasPort.P4_Fwd;
else if (port.ToUpper() == "P5_FWD") coupledPort.DUT_P5_P8 = MeasPort.P5_Fwd;
else if (port.ToUpper() == "P6_FWD") coupledPort.DUT_P5_P8 = MeasPort.P6_Fwd;
else if (port.ToUpper() == "P7_FWD") coupledPort.DUT_P5_P8 = MeasPort.P7_Fwd;
else if (port.ToUpper() == "P8_FWD") coupledPort.DUT_P5_P8 = MeasPort.P8_Fwd;

else if (port.ToUpper() == "P1_REV") coupledPort.DUT_P5_P8 = MeasPort.P1_Rev;
else if (port.ToUpper() == "P2_REV") coupledPort.DUT_P5_P8 = MeasPort.P2_Rev;
else if (port.ToUpper() == "P3_REV") coupledPort.DUT_P5_P8 = MeasPort.P3_Rev;
else if (port.ToUpper() == "P4_REV") coupledPort.DUT_P5_P8 = MeasPort.P4_Rev;
else if (port.ToUpper() == "P5_REV") coupledPort.DUT_P5_P8 = MeasPort.P5_Rev;
else if (port.ToUpper() == "P6_REV") coupledPort.DUT_P5_P8 = MeasPort.P6_Rev;
else if (port.ToUpper() == "P7_REV") coupledPort.DUT_P5_P8 = MeasPort.P7_Rev;
else if (port.ToUpper() == "P8_REV") coupledPort.DUT_P5_P8 = MeasPort.P8_Rev;

else if (port.ToUpper() == "M1") coupledPort.DUT_P5_P8 = MeasPort.M1;
else if (port.ToUpper() == "M2") coupledPort.DUT_P5_P8 = MeasPort.M2;
else if (port.ToUpper() == "M3") coupledPort.DUT_P5_P8 = MeasPort.M3;
else if (port.ToUpper() == "M4") coupledPort.DUT_P5_P8 = MeasPort.M4;
else if (port.ToUpper() == "M5") coupledPort.DUT_P5_P8 = MeasPort.M5;
else if (port.ToUpper() == "M6") coupledPort.DUT_P5_P8 = MeasPort.M6;
```

```
else if (port.ToUpper() == "M7") coupledPort.DUT_P5_P8 = MeasPort.M7;
else if (port.ToUpper() == "M8") coupledPort.DUT_P5_P8 = MeasPort.M8;
else if (port.ToUpper() == "M9") coupledPort.DUT_P5_P8 = MeasPort.M9;
else if (port.ToUpper() == "M10") coupledPort.DUT_P5_P8 = MeasPort.M10;
else if (port.ToUpper() == "M11") coupledPort.DUT_P5_P8 = MeasPort.M11;
else if (port.ToUpper() == "M12") coupledPort.DUT_P5_P8 = MeasPort.M12;
else if (port.ToUpper() == "M13") coupledPort.DUT_P5_P8 = MeasPort.M13;
else if (port.ToUpper() == "M14") coupledPort.DUT_P5_P8 = MeasPort.M14;
else if (port.ToUpper() == "M15") coupledPort.DUT_P5_P8 = MeasPort.M15;
else if (port.ToUpper() == "M16") coupledPort.DUT_P5_P8 = MeasPort.M16;
else if (port.ToUpper() == "M17") coupledPort.DUT_P5_P8 = MeasPort.M17;
else if (port.ToUpper() == "M18") coupledPort.DUT_P5_P8 = MeasPort.M18;
else if (port.ToUpper() == "M19") coupledPort.DUT_P5_P8 = MeasPort.M19;
else if (port.ToUpper() == "M20") coupledPort.DUT_P5_P8 = MeasPort.M20;
else if (port.ToUpper() == "M21") coupledPort.DUT_P5_P8 = MeasPort.M21;
else if (port.ToUpper() == "M22") coupledPort.DUT_P5_P8 = MeasPort.M22;
else if (port.ToUpper() == "M23") coupledPort.DUT_P5_P8 = MeasPort.M23;
else if (port.ToUpper() == "M24") coupledPort.DUT_P5_P8 = MeasPort.M24;
else if (port.ToUpper() == "M25") coupledPort.DUT_P5_P8 = MeasPort.M25;
else if (port.ToUpper() == "M26") coupledPort.DUT_P5_P8 = MeasPort.M26;

else if (port.ToUpper() == "MA") coupledPort.DUT_P5_P8 = MeasPort.MA;
else if (port.ToUpper() == "MB") coupledPort.DUT_P5_P8 = MeasPort.MB;
else if (port.ToUpper() == "MC") coupledPort.DUT_P5_P8 = MeasPort.MC;
else if (port.ToUpper() == "MD") coupledPort.DUT_P5_P8 = MeasPort.MD;

else coupledPort.DUT_P5_P8 = MeasPort.NONE;
}
else if (tmpString.ToUpper() == "SWITCH CONFIGURATION:")
{
string switchConfig = "";

line = stream.ReadLine();

for (int i = 0; i < 8; i++)
{
switchConfig = line.Split(',')[i + 1];

if (switchConfig.ToUpper() == "BB") switchSelect.switchPosition[i] =
SwitchSelect.SwitchPosition.BB;
else if (switchConfig.ToUpper() == "FILT") switchSelect.switchPosition[i] =
SwitchSelect.SwitchPosition.FILT;
```

```
else if (switchConfig.ToUpper() == "HB") switchSelect.switchPosition[i] =
SwitchSelect.SwitchPosition.HB;
else if (switchConfig.ToUpper() == "HB_DESELECT") switchSelect.switchPosition[i] =
SwitchSelect.SwitchPosition.HB_Deselect;
else if (switchConfig.ToUpper() == "LB_HPF") switchSelect.switchPosition[i] =
SwitchSelect.SwitchPosition.LB_HPF;
else switchSelect.switchPosition[i] = SwitchSelect.SwitchPosition.None;
}
}
else if (tmpString.ToUpper() == "PORT MODULE ATTENUATION:")
{
line = stream.ReadLine();

for (int i = 0; i < 17; i++)
{
atten.srcAtten[i] = double.Parse(line.Split(',')[i + 1]);
}
for (int i = 0; i < 30; i++)
{
atten.measAtten[i] = double.Parse(line.Split(',')[i + 18]);
}
}
else if (tmpString.ToUpper() == "SOURCE")
{
while ((line = stream.ReadLine()) != null)
{
string tmp = "";

// Source
tmp = line.Split(',')[0];
if (tmp.ToUpper() == "SG1") sigGen = SigGen.SG1;
else if (tmp.ToUpper() == "SG2") sigGen = SigGen.SG2;
else if (tmp.ToUpper() == "SG1_SG2") sigGen = SigGen.SG1_SG2;
else if (tmp.ToUpper() == "NOISE") sigGen = SigGen.NOISE;
else sigGen = SigGen.ISOLATE;

// Source Port
tmp = line.Split(',')[1];
if (tmp.ToUpper() == "P1") srcPort = SrcPort.P1;
else if (tmp.ToUpper() == "P2") srcPort = SrcPort.P2;
else if (tmp.ToUpper() == "P3") srcPort = SrcPort.P3;
else if (tmp.ToUpper() == "P4") srcPort = SrcPort.P4;
```

```
else if (tmp.ToUpper() == "P5") srcPort = SrcPort.P5;
else if (tmp.ToUpper() == "P6") srcPort = SrcPort.P6;
else if (tmp.ToUpper() == "P7") srcPort = SrcPort.P7;
else if (tmp.ToUpper() == "P8") srcPort = SrcPort.P8;
else if (tmp.ToUpper() == "P9") srcPort = SrcPort.P9;
else if (tmp.ToUpper() == "P10") srcPort = SrcPort.P10;
else if (tmp.ToUpper() == "P11") srcPort = SrcPort.P11;
else if (tmp.ToUpper() == "P12") srcPort = SrcPort.P12;
else if (tmp.ToUpper() == "P13") srcPort = SrcPort.P13;
else if (tmp.ToUpper() == "P14") srcPort = SrcPort.P14;
else if (tmp.ToUpper() == "P15") srcPort = SrcPort.P15;
else if (tmp.ToUpper() == "P16") srcPort = SrcPort.P16;
else if (tmp.ToUpper() == "CAL") srcPort = SrcPort.CAL;
else if (tmp.ToUpper() == "SG1_OUT") srcPort = SrcPort.SG1_OUT;
else srcPort = SrcPort.NONE;

// Source Port Alias
srcPortAlias = line.Split(',')[2];

// Amp Port
tmp = line.Split(',')[3];
if (tmp.ToUpper() == "HPA1") ampPort = CalDataRF300.AmpPort.HPA1;
else if (tmp.ToUpper() == "HPA2") ampPort = CalDataRF300.AmpPort.HPA2;
else if (tmp.ToUpper() == "HPA3") ampPort = CalDataRF300.AmpPort.HPA3;
else if (tmp.ToUpper() == "HPA4") ampPort = CalDataRF300.AmpPort.HPA4;
else ampPort = CalDataRF300.AmpPort.None;

// Source RF300 Primary
tmp = line.Split(',')[4];
if (tmp.ToUpper() == "DUT_P1") srcPrimary = CalDataRF300.sfPort.DUT_P1;
else if (tmp.ToUpper() == "DUT_P1_FWD") srcPrimary = CalDataRF300.sfPort.DUT_P1_FWD;
else if (tmp.ToUpper() == "DUT_P1_REV") srcPrimary = CalDataRF300.sfPort.DUT_P1_REV;
else if (tmp.ToUpper() == "DUT_P2") srcPrimary = CalDataRF300.sfPort.DUT_P2;
else if (tmp.ToUpper() == "DUT_P2_FWD") srcPrimary = CalDataRF300.sfPort.DUT_P2_FWD;
else if (tmp.ToUpper() == "DUT_P2_REV") srcPrimary = CalDataRF300.sfPort.DUT_P2_REV;
else if (tmp.ToUpper() == "DUT_P3") srcPrimary = CalDataRF300.sfPort.DUT_P3;
else if (tmp.ToUpper() == "DUT_P3_FWD") srcPrimary = CalDataRF300.sfPort.DUT_P3_FWD;
else if (tmp.ToUpper() == "DUT_P3_REV") srcPrimary = CalDataRF300.sfPort.DUT_P3_REV;
else if (tmp.ToUpper() == "DUT_P4") srcPrimary = CalDataRF300.sfPort.DUT_P4;
else if (tmp.ToUpper() == "DUT_P4_FWD") srcPrimary = CalDataRF300.sfPort.DUT_P4_FWD;
else if (tmp.ToUpper() == "DUT_P4_REV") srcPrimary = CalDataRF300.sfPort.DUT_P4_REV;
else if (tmp.ToUpper() == "DUT_P5") srcPrimary = CalDataRF300.sfPort.DUT_P5;
```

```
else if (tmp.ToUpper() == "DUT_P5_FWD") srcPrimary = CalDataRF300.sfPort.DUT_P5_FWD;
else if (tmp.ToUpper() == "DUT_P5_REV") srcPrimary = CalDataRF300.sfPort.DUT_P5_REV;
else if (tmp.ToUpper() == "DUT_P6") srcPrimary = CalDataRF300.sfPort.DUT_P6;
else if (tmp.ToUpper() == "DUT_P6_FWD") srcPrimary = CalDataRF300.sfPort.DUT_P6_FWD;
else if (tmp.ToUpper() == "DUT_P6_REV") srcPrimary = CalDataRF300.sfPort.DUT_P6_REV;
else if (tmp.ToUpper() == "DUT_P7") srcPrimary = CalDataRF300.sfPort.DUT_P7;
else if (tmp.ToUpper() == "DUT_P7_FWD") srcPrimary = CalDataRF300.sfPort.DUT_P7_FWD;
else if (tmp.ToUpper() == "DUT_P7_REV") srcPrimary = CalDataRF300.sfPort.DUT_P7_REV;
else if (tmp.ToUpper() == "DUT_P8") srcPrimary = CalDataRF300.sfPort.DUT_P8;
else if (tmp.ToUpper() == "DUT_P8_FWD") srcPrimary = CalDataRF300.sfPort.DUT_P8_FWD;
else if (tmp.ToUpper() == "DUT_P8_REV") srcPrimary = CalDataRF300.sfPort.DUT_P8_REV;
else if (tmp.ToUpper() == "DUT_P1_P4_COUPLED") srcPrimary =
CalDataRF300.sfPort.DUT_P1_P4_COUPLED;
else if (tmp.ToUpper() == "DUT_P5_P8_COUPLED") srcPrimary =
CalDataRF300.sfPort.DUT_P5_P8_COUPLED;
else if (tmp.ToUpper() == "BB1") srcPrimary = CalDataRF300.sfPort.BB1;
else if (tmp.ToUpper() == "BB2") srcPrimary = CalDataRF300.sfPort.BB2;
else if (tmp.ToUpper() == "BB3") srcPrimary = CalDataRF300.sfPort.BB3;
else if (tmp.ToUpper() == "BB4") srcPrimary = CalDataRF300.sfPort.BB4;
else if (tmp.ToUpper() == "HB1_HPF") srcPrimary = CalDataRF300.sfPort.HB1_HPF;
else if (tmp.ToUpper() == "HB1_LPF") srcPrimary = CalDataRF300.sfPort.HB1_LPF;
else if (tmp.ToUpper() == "LB1_HPF") srcPrimary = CalDataRF300.sfPort.LB1_HPF;
else if (tmp.ToUpper() == "LB1_LPF") srcPrimary = CalDataRF300.sfPort.LB1_LPF;
else if (tmp.ToUpper() == "HB2_HPF") srcPrimary = CalDataRF300.sfPort.HB2_HPF;
else if (tmp.ToUpper() == "HB2_LPF") srcPrimary = CalDataRF300.sfPort.HB2_LPF;
else if (tmp.ToUpper() == "LB2_HPF") srcPrimary = CalDataRF300.sfPort.LB2_HPF;
else if (tmp.ToUpper() == "LB2_LPF") srcPrimary = CalDataRF300.sfPort.LB2_LPF;
else if (tmp.ToUpper() == "DM1") srcPrimary = CalDataRF300.sfPort.DM1;
else if (tmp.ToUpper() == "DM2") srcPrimary = CalDataRF300.sfPort.DM2;
else if (tmp.ToUpper() == "DM3") srcPrimary = CalDataRF300.sfPort.DM3;
else if (tmp.ToUpper() == "DM4") srcPrimary = CalDataRF300.sfPort.DM4;
else srcPrimary = CalDataRF300.sfPort.None;

// Source RF300 Secondary
tmp = line.Split(',')[5];
if (tmp.ToUpper() == "DUT_P1") srcSecondary = CalDataRF300.sfPort.DUT_P1;
else if (tmp.ToUpper() == "DUT_P1_FWD") srcSecondary = CalDataRF300.sfPort.DUT_P1_FWD;
else if (tmp.ToUpper() == "DUT_P1_REV") srcSecondary = CalDataRF300.sfPort.DUT_P1_REV;
else if (tmp.ToUpper() == "DUT_P2") srcSecondary = CalDataRF300.sfPort.DUT_P2;
else if (tmp.ToUpper() == "DUT_P2_FWD") srcSecondary = CalDataRF300.sfPort.DUT_P2_FWD;
else if (tmp.ToUpper() == "DUT_P2_REV") srcSecondary = CalDataRF300.sfPort.DUT_P2_REV;
else if (tmp.ToUpper() == "DUT_P3") srcSecondary = CalDataRF300.sfPort.DUT_P3;
```

```
else if (tmp.ToUpper() == "DUT_P3_FWD") srcSecondary = CalDataRF300.sfPort.DUT_P3_FWD;
else if (tmp.ToUpper() == "DUT_P3_REV") srcSecondary = CalDataRF300.sfPort.DUT_P3_REV;
else if (tmp.ToUpper() == "DUT_P4") srcSecondary = CalDataRF300.sfPort.DUT_P4;
else if (tmp.ToUpper() == "DUT_P4_FWD") srcSecondary = CalDataRF300.sfPort.DUT_P4_FWD;
else if (tmp.ToUpper() == "DUT_P4_REV") srcSecondary = CalDataRF300.sfPort.DUT_P4_REV;
else if (tmp.ToUpper() == "DUT_P5") srcSecondary = CalDataRF300.sfPort.DUT_P5;
else if (tmp.ToUpper() == "DUT_P5_FWD") srcSecondary = CalDataRF300.sfPort.DUT_P5_FWD;
else if (tmp.ToUpper() == "DUT_P5_REV") srcSecondary = CalDataRF300.sfPort.DUT_P5_REV;
else if (tmp.ToUpper() == "DUT_P6") srcSecondary = CalDataRF300.sfPort.DUT_P6;
else if (tmp.ToUpper() == "DUT_P6_FWD") srcSecondary = CalDataRF300.sfPort.DUT_P6_FWD;
else if (tmp.ToUpper() == "DUT_P6_REV") srcSecondary = CalDataRF300.sfPort.DUT_P6_REV;
else if (tmp.ToUpper() == "DUT_P7") srcSecondary = CalDataRF300.sfPort.DUT_P7;
else if (tmp.ToUpper() == "DUT_P7_FWD") srcSecondary = CalDataRF300.sfPort.DUT_P7_FWD;
else if (tmp.ToUpper() == "DUT_P7_REV") srcSecondary = CalDataRF300.sfPort.DUT_P7_REV;
else if (tmp.ToUpper() == "DUT_P8") srcSecondary = CalDataRF300.sfPort.DUT_P8;
else if (tmp.ToUpper() == "DUT_P8_FWD") srcSecondary = CalDataRF300.sfPort.DUT_P8_FWD;
else if (tmp.ToUpper() == "DUT_P8_REV") srcSecondary = CalDataRF300.sfPort.DUT_P8_REV;
else if (tmp.ToUpper() == "DUT_P1_P4_COUPLED") srcSecondary =
CalDataRF300.sfPort.DUT_P1_P4_COUPLED;
else if (tmp.ToUpper() == "DUT_P5_P8_COUPLED") srcSecondary =
CalDataRF300.sfPort.DUT_P5_P8_COUPLED;
else if (tmp.ToUpper() == "BB1") srcSecondary = CalDataRF300.sfPort.BB1;
else if (tmp.ToUpper() == "BB2") srcSecondary = CalDataRF300.sfPort.BB2;
else if (tmp.ToUpper() == "BB3") srcSecondary = CalDataRF300.sfPort.BB3;
else if (tmp.ToUpper() == "BB4") srcSecondary = CalDataRF300.sfPort.BB4;
else if (tmp.ToUpper() == "HB1_HPF") srcSecondary = CalDataRF300.sfPort.HB1_HPF;
else if (tmp.ToUpper() == "HB1_LPF") srcSecondary = CalDataRF300.sfPort.HB1_LPF;
else if (tmp.ToUpper() == "LB1_HPF") srcSecondary = CalDataRF300.sfPort.LB1_HPF;
else if (tmp.ToUpper() == "LB1_LPF") srcSecondary = CalDataRF300.sfPort.LB1_LPF;
else if (tmp.ToUpper() == "HB2_HPF") srcSecondary = CalDataRF300.sfPort.HB2_HPF;
else if (tmp.ToUpper() == "HB2_LPF") srcSecondary = CalDataRF300.sfPort.HB2_LPF;
else if (tmp.ToUpper() == "LB2_HPF") srcSecondary = CalDataRF300.sfPort.LB2_HPF;
else if (tmp.ToUpper() == "LB2_LPF") srcSecondary = CalDataRF300.sfPort.LB2_LPF;
else if (tmp.ToUpper() == "DM1") srcSecondary = CalDataRF300.sfPort.DM1;
else if (tmp.ToUpper() == "DM2") srcSecondary = CalDataRF300.sfPort.DM2;
else if (tmp.ToUpper() == "DM3") srcSecondary = CalDataRF300.sfPort.DM3;
else if (tmp.ToUpper() == "DM4") srcSecondary = CalDataRF300.sfPort.DM4;
else srcSecondary = CalDataRF300.sfPort.None;

// Source Coupled Port
tmp = line.Split(',')[6];
if (tmp.ToUpper() == "DUT_P1_FWD") srcCoupler = CalDataRF300.sfCouplerPort.DUT_P1_FWD;
```

```
else if (tmp.ToUpper() == "DUT_P1_REV") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P1_REV;
else if (tmp.ToUpper() == "DUT_P2_FWD") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P2_FWD;
else if (tmp.ToUpper() == "DUT_P2_REV") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P2_REV;
else if (tmp.ToUpper() == "DUT_P3_FWD") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P3_FWD;
else if (tmp.ToUpper() == "DUT_P3_REV") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P3_REV;
else if (tmp.ToUpper() == "DUT_P4_FWD") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P4_FWD;
else if (tmp.ToUpper() == "DUT_P4_REV") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P4_REV;
else if (tmp.ToUpper() == "DUT_P5_FWD") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P5_FWD;
else if (tmp.ToUpper() == "DUT_P5_REV") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P5_REV;
else if (tmp.ToUpper() == "DUT_P6_FWD") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P6_FWD;
else if (tmp.ToUpper() == "DUT_P6_REV") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P6_REV;
else if (tmp.ToUpper() == "DUT_P7_FWD") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P7_FWD;
else if (tmp.ToUpper() == "DUT_P7_REV") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P7_REV;
else if (tmp.ToUpper() == "DUT_P8_FWD") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P8_FWD;
else if (tmp.ToUpper() == "DUT_P8_REV") srcCoupler =
CalDataRF300.sfCouplerPort.DUT_P8_REV;
else srcCoupler = CalDataRF300.sfCouplerPort.None;

// Measure Source Cal Path
//tmp = line.Split(',')[7];

// Measure RF300 Primary
tmp = line.Split(',')[8];
if (tmp.ToUpper() == "DUT_P1") measPrimary = CalDataRF300.sfPort.DUT_P1;
else if (tmp.ToUpper() == "DUT_P1_FWD") measPrimary = CalDataRF300.sfPort.DUT_P1_FWD;
else if (tmp.ToUpper() == "DUT_P1_REV") measPrimary = CalDataRF300.sfPort.DUT_P1_REV;
else if (tmp.ToUpper() == "DUT_P2") measPrimary = CalDataRF300.sfPort.DUT_P2;
else if (tmp.ToUpper() == "DUT_P2_FWD") measPrimary = CalDataRF300.sfPort.DUT_P2_FWD;
```

```
else if (tmp.ToUpper() == "DUT_P2_REV") measPrimary = CalDataRF300.sfPort.DUT_P2_REV;
else if (tmp.ToUpper() == "DUT_P3") measPrimary = CalDataRF300.sfPort.DUT_P3;
else if (tmp.ToUpper() == "DUT_P3_FWD") measPrimary = CalDataRF300.sfPort.DUT_P3_FWD;
else if (tmp.ToUpper() == "DUT_P3_REV") measPrimary = CalDataRF300.sfPort.DUT_P3_REV;
else if (tmp.ToUpper() == "DUT_P4") measPrimary = CalDataRF300.sfPort.DUT_P4;
else if (tmp.ToUpper() == "DUT_P4_FWD") measPrimary = CalDataRF300.sfPort.DUT_P4_FWD;
else if (tmp.ToUpper() == "DUT_P4_REV") measPrimary = CalDataRF300.sfPort.DUT_P4_REV;
else if (tmp.ToUpper() == "DUT_P5") measPrimary = CalDataRF300.sfPort.DUT_P5;
else if (tmp.ToUpper() == "DUT_P5_FWD") measPrimary = CalDataRF300.sfPort.DUT_P5_FWD;
else if (tmp.ToUpper() == "DUT_P5_REV") measPrimary = CalDataRF300.sfPort.DUT_P5_REV;
else if (tmp.ToUpper() == "DUT_P6") measPrimary = CalDataRF300.sfPort.DUT_P6;
else if (tmp.ToUpper() == "DUT_P6_FWD") measPrimary = CalDataRF300.sfPort.DUT_P6_FWD;
else if (tmp.ToUpper() == "DUT_P6_REV") measPrimary = CalDataRF300.sfPort.DUT_P6_REV;
else if (tmp.ToUpper() == "DUT_P7") measPrimary = CalDataRF300.sfPort.DUT_P7;
else if (tmp.ToUpper() == "DUT_P7_FWD") measPrimary = CalDataRF300.sfPort.DUT_P7_FWD;
else if (tmp.ToUpper() == "DUT_P7_REV") measPrimary = CalDataRF300.sfPort.DUT_P7_REV;
else if (tmp.ToUpper() == "DUT_P8") measPrimary = CalDataRF300.sfPort.DUT_P8;
else if (tmp.ToUpper() == "DUT_P8_FWD") measPrimary = CalDataRF300.sfPort.DUT_P8_FWD;
else if (tmp.ToUpper() == "DUT_P8_REV") measPrimary = CalDataRF300.sfPort.DUT_P8_REV;
else if (tmp.ToUpper() == "DUT_P1_P4_COUPLED") measPrimary =
CalDataRF300.sfPort.DUT_P1_P4_COUPLED;
else if (tmp.ToUpper() == "DUT_P5_P8_COUPLED") measPrimary =
CalDataRF300.sfPort.DUT_P5_P8_COUPLED;
else if (tmp.ToUpper() == "BB1") measPrimary = CalDataRF300.sfPort.BB1;
else if (tmp.ToUpper() == "BB2") measPrimary = CalDataRF300.sfPort.BB2;
else if (tmp.ToUpper() == "BB3") measPrimary = CalDataRF300.sfPort.BB3;
else if (tmp.ToUpper() == "BB4") measPrimary = CalDataRF300.sfPort.BB4;
else if (tmp.ToUpper() == "HB1_HPF") measPrimary = CalDataRF300.sfPort.HB1_HPF;
else if (tmp.ToUpper() == "HB1_LPF") measPrimary = CalDataRF300.sfPort.HB1_LPF;
else if (tmp.ToUpper() == "LB1_HPF") measPrimary = CalDataRF300.sfPort.LB1_HPF;
else if (tmp.ToUpper() == "LB1_LPF") measPrimary = CalDataRF300.sfPort.LB1_LPF;
else if (tmp.ToUpper() == "HB2_HPF") measPrimary = CalDataRF300.sfPort.HB2_HPF;
else if (tmp.ToUpper() == "HB2_LPF") measPrimary = CalDataRF300.sfPort.HB2_LPF;
else if (tmp.ToUpper() == "LB2_HPF") measPrimary = CalDataRF300.sfPort.LB2_HPF;
else if (tmp.ToUpper() == "LB2_LPF") measPrimary = CalDataRF300.sfPort.LB2_LPF;
else if (tmp.ToUpper() == "DM1") measPrimary = CalDataRF300.sfPort.DM1;
else if (tmp.ToUpper() == "DM2") measPrimary = CalDataRF300.sfPort.DM2;
else if (tmp.ToUpper() == "DM3") measPrimary = CalDataRF300.sfPort.DM3;
else if (tmp.ToUpper() == "DM4") measPrimary = CalDataRF300.sfPort.DM4;
else measPrimary = CalDataRF300.sfPort.None;

// Measure RF300 Secondary
```

```
tmp = line.Split(',')[9];
if (tmp.ToUpper() == "DUT_P1") measSecondary = CalDataRF300.sfPort.DUT_P1;
else if (tmp.ToUpper() == "DUT_P1_FWD") measSecondary =
CalDataRF300.sfPort.DUT_P1_FWD;
else if (tmp.ToUpper() == "DUT_P1_REV") measSecondary =
CalDataRF300.sfPort.DUT_P1_REV;
else if (tmp.ToUpper() == "DUT_P2") measSecondary = CalDataRF300.sfPort.DUT_P2;
else if (tmp.ToUpper() == "DUT_P2_FWD") measSecondary =
CalDataRF300.sfPort.DUT_P2_FWD;
else if (tmp.ToUpper() == "DUT_P2_REV") measSecondary =
CalDataRF300.sfPort.DUT_P2_REV;
else if (tmp.ToUpper() == "DUT_P3") measSecondary = CalDataRF300.sfPort.DUT_P3;
else if (tmp.ToUpper() == "DUT_P3_FWD") measSecondary =
CalDataRF300.sfPort.DUT_P3_FWD;
else if (tmp.ToUpper() == "DUT_P3_REV") measSecondary =
CalDataRF300.sfPort.DUT_P3_REV;
else if (tmp.ToUpper() == "DUT_P4") measSecondary = CalDataRF300.sfPort.DUT_P4;
else if (tmp.ToUpper() == "DUT_P4_FWD") measSecondary =
CalDataRF300.sfPort.DUT_P4_FWD;
else if (tmp.ToUpper() == "DUT_P4_REV") measSecondary =
CalDataRF300.sfPort.DUT_P4_REV;
else if (tmp.ToUpper() == "DUT_P5") measSecondary = CalDataRF300.sfPort.DUT_P5;
else if (tmp.ToUpper() == "DUT_P5_FWD") measSecondary =
CalDataRF300.sfPort.DUT_P5_FWD;
else if (tmp.ToUpper() == "DUT_P5_REV") measSecondary =
CalDataRF300.sfPort.DUT_P5_REV;
else if (tmp.ToUpper() == "DUT_P6") measSecondary = CalDataRF300.sfPort.DUT_P6;
else if (tmp.ToUpper() == "DUT_P6_FWD") measSecondary =
CalDataRF300.sfPort.DUT_P6_FWD;
else if (tmp.ToUpper() == "DUT_P6_REV") measSecondary =
CalDataRF300.sfPort.DUT_P6_REV;
else if (tmp.ToUpper() == "DUT_P7") measSecondary = CalDataRF300.sfPort.DUT_P7;
else if (tmp.ToUpper() == "DUT_P7_FWD") measSecondary =
CalDataRF300.sfPort.DUT_P7_FWD;
else if (tmp.ToUpper() == "DUT_P7_REV") measSecondary =
CalDataRF300.sfPort.DUT_P7_REV;
else if (tmp.ToUpper() == "DUT_P8") measSecondary = CalDataRF300.sfPort.DUT_P8;
else if (tmp.ToUpper() == "DUT_P8_FWD") measSecondary =
CalDataRF300.sfPort.DUT_P8_FWD;
else if (tmp.ToUpper() == "DUT_P8_REV") measSecondary =
CalDataRF300.sfPort.DUT_P8_REV;
else if (tmp.ToUpper() == "DUT_P1_P4_COUPLED") measSecondary =
```

```
CalDataRF300.sfPort.DUT_P1_P4_COUPLED;
else if (tmp.ToUpper() == "DUT_P5_P8_COUPLED") measSecondary =
CalDataRF300.sfPort.DUT_P5_P8_COUPLED;
else if (tmp.ToUpper() == "BB1") measSecondary = CalDataRF300.sfPort.BB1;
else if (tmp.ToUpper() == "BB2") measSecondary = CalDataRF300.sfPort.BB2;
else if (tmp.ToUpper() == "BB3") measSecondary = CalDataRF300.sfPort.BB3;
else if (tmp.ToUpper() == "BB4") measSecondary = CalDataRF300.sfPort.BB4;
else if (tmp.ToUpper() == "HB1_HPF") measSecondary = CalDataRF300.sfPort.HB1_HPF;
else if (tmp.ToUpper() == "HB1_LPF") measSecondary = CalDataRF300.sfPort.HB1_LPF;
else if (tmp.ToUpper() == "LB1_HPF") measSecondary = CalDataRF300.sfPort.LB1_HPF;
else if (tmp.ToUpper() == "LB1_LPF") measSecondary = CalDataRF300.sfPort.LB1_LPF;
else if (tmp.ToUpper() == "HB2_HPF") measSecondary = CalDataRF300.sfPort.HB2_HPF;
else if (tmp.ToUpper() == "HB2_LPF") measSecondary = CalDataRF300.sfPort.HB2_LPF;
else if (tmp.ToUpper() == "LB2_HPF") measSecondary = CalDataRF300.sfPort.LB2_HPF;
else if (tmp.ToUpper() == "LB2_LPF") measSecondary = CalDataRF300.sfPort.LB2_LPF;
else if (tmp.ToUpper() == "DM1") measSecondary = CalDataRF300.sfPort.DM1;
else if (tmp.ToUpper() == "DM2") measSecondary = CalDataRF300.sfPort.DM2;
else if (tmp.ToUpper() == "DM3") measSecondary = CalDataRF300.sfPort.DM3;
else if (tmp.ToUpper() == "DM4") measSecondary = CalDataRF300.sfPort.DM4;
else measSecondary = CalDataRF300.sfPort.None;

// Measure Port
tmp = line.Split(',')[10];
if (tmp.ToUpper() == "P1") measPort = MeasPort.P1;
else if (tmp.ToUpper() == "P2") measPort = MeasPort.P2;
else if (tmp.ToUpper() == "P3") measPort = MeasPort.P3;
else if (tmp.ToUpper() == "P4") measPort = MeasPort.P4;
else if (tmp.ToUpper() == "P5") measPort = MeasPort.P5;
else if (tmp.ToUpper() == "P6") measPort = MeasPort.P6;
else if (tmp.ToUpper() == "P7") measPort = MeasPort.P7;
else if (tmp.ToUpper() == "P8") measPort = MeasPort.P8;
else if (tmp.ToUpper() == "P9") measPort = MeasPort.P9;
else if (tmp.ToUpper() == "P10") measPort = MeasPort.P10;
else if (tmp.ToUpper() == "P11") measPort = MeasPort.P11;
else if (tmp.ToUpper() == "P12") measPort = MeasPort.P12;
else if (tmp.ToUpper() == "P13") measPort = MeasPort.P13;
else if (tmp.ToUpper() == "P14") measPort = MeasPort.P14;
else if (tmp.ToUpper() == "P15") measPort = MeasPort.P15;
else if (tmp.ToUpper() == "P16") measPort = MeasPort.P16;
else if (tmp.ToUpper() == "P1_FWD") measPort = MeasPort.P1_Fwd;
else if (tmp.ToUpper() == "P2_FWD") measPort = MeasPort.P2_Fwd;
else if (tmp.ToUpper() == "P3_FWD") measPort = MeasPort.P3_Fwd;
```

```
else if (tmp.ToUpper() == "P4_FWD") measPort = MeasPort.P4_Fwd;
else if (tmp.ToUpper() == "P5_FWD") measPort = MeasPort.P5_Fwd;
else if (tmp.ToUpper() == "P6_FWD") measPort = MeasPort.P6_Fwd;
else if (tmp.ToUpper() == "P7_FWD") measPort = MeasPort.P7_Fwd;
else if (tmp.ToUpper() == "P8_FWD") measPort = MeasPort.P8_Fwd;
else if (tmp.ToUpper() == "P1_REV") measPort = MeasPort.P1_Rev;
else if (tmp.ToUpper() == "P2_REV") measPort = MeasPort.P2_Rev;
else if (tmp.ToUpper() == "P3_REV") measPort = MeasPort.P3_Rev;
else if (tmp.ToUpper() == "P4_REV") measPort = MeasPort.P4_Rev;
else if (tmp.ToUpper() == "P5_REV") measPort = MeasPort.P5_Rev;
else if (tmp.ToUpper() == "P6_REV") measPort = MeasPort.P6_Rev;
else if (tmp.ToUpper() == "P7_REV") measPort = MeasPort.P7_Rev;
else if (tmp.ToUpper() == "P8_REV") measPort = MeasPort.P8_Rev;
else if (tmp.ToUpper() == "M1") measPort = MeasPort.M1;
else if (tmp.ToUpper() == "M2") measPort = MeasPort.M2;
else if (tmp.ToUpper() == "M3") measPort = MeasPort.M3;
else if (tmp.ToUpper() == "M4") measPort = MeasPort.M4;
else if (tmp.ToUpper() == "M5") measPort = MeasPort.M5;
else if (tmp.ToUpper() == "M6") measPort = MeasPort.M6;
else if (tmp.ToUpper() == "M7") measPort = MeasPort.M7;
else if (tmp.ToUpper() == "M8") measPort = MeasPort.M8;
else if (tmp.ToUpper() == "M9") measPort = MeasPort.M9;
else if (tmp.ToUpper() == "M10") measPort = MeasPort.M10;
else if (tmp.ToUpper() == "M11") measPort = MeasPort.M11;
else if (tmp.ToUpper() == "M12") measPort = MeasPort.M12;
else if (tmp.ToUpper() == "M13") measPort = MeasPort.M13;
else if (tmp.ToUpper() == "M14") measPort = MeasPort.M14;
else if (tmp.ToUpper() == "M15") measPort = MeasPort.M15;
else if (tmp.ToUpper() == "M16") measPort = MeasPort.M16;
else if (tmp.ToUpper() == "M17") measPort = MeasPort.M17;
else if (tmp.ToUpper() == "M18") measPort = MeasPort.M18;
else if (tmp.ToUpper() == "M19") measPort = MeasPort.M19;
else if (tmp.ToUpper() == "M20") measPort = MeasPort.M20;
else if (tmp.ToUpper() == "M21") measPort = MeasPort.M21;
else if (tmp.ToUpper() == "M22") measPort = MeasPort.M22;
else if (tmp.ToUpper() == "M23") measPort = MeasPort.M23;
else if (tmp.ToUpper() == "M24") measPort = MeasPort.M24;
else if (tmp.ToUpper() == "M25") measPort = MeasPort.M25;
else if (tmp.ToUpper() == "M26") measPort = MeasPort.M26;
else if (tmp.ToUpper() == "MA") measPort = MeasPort.MA;
else if (tmp.ToUpper() == "MB") measPort = MeasPort.MB;
else if (tmp.ToUpper() == "MC") measPort = MeasPort.MC;
```

```
else if (tmp.ToUpper() == "MD") measPort = MeasPort.MD;
else measPort = MeasPort.NONE;

// Source Port Alias
measPortAlias = line.Split(',')[11];

// Measure Coupled Port
tmp = line.Split(',')[12];
if (tmp.ToUpper() == "DUT_P1_FWD") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P1_FWD;
else if (tmp.ToUpper() == "DUT_P1_REV") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P1_REV;
else if (tmp.ToUpper() == "DUT_P2_FWD") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P2_FWD;
else if (tmp.ToUpper() == "DUT_P2_REV") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P2_REV;
else if (tmp.ToUpper() == "DUT_P3_FWD") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P3_FWD;
else if (tmp.ToUpper() == "DUT_P3_REV") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P3_REV;
else if (tmp.ToUpper() == "DUT_P4_FWD") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P4_FWD;
else if (tmp.ToUpper() == "DUT_P4_REV") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P4_REV;
else if (tmp.ToUpper() == "DUT_P5_FWD") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P5_FWD;
else if (tmp.ToUpper() == "DUT_P5_REV") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P5_REV;
else if (tmp.ToUpper() == "DUT_P6_FWD") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P6_FWD;
else if (tmp.ToUpper() == "DUT_P6_REV") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P6_REV;
else if (tmp.ToUpper() == "DUT_P7_FWD") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P7_FWD;
else if (tmp.ToUpper() == "DUT_P7_REV") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P7_REV;
else if (tmp.ToUpper() == "DUT_P8_FWD") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P8_FWD;
else if (tmp.ToUpper() == "DUT_P8_REV") measCoupler =
CalDataRF300.sfCouplerPort.DUT_P8_REV;
else measCoupler = CalDataRF300.sfCouplerPort.None;
```

```
// Frequency
freq = double.Parse(line.Split(',')[13]);

// Level
level = double.Parse(line.Split(',')[14]);

// Filter
tmp = line.Split(',')[15];
if (tmp.ToUpper() == "BYPASS") filter = MeasFilt.Bypass;
else if (tmp.ToUpper() == "ATTEN") filter = MeasFilt.Atten;
else if (tmp.ToUpper() == "FILT1") filter = MeasFilt.FILT1;
else if (tmp.ToUpper() == "FILT2") filter = MeasFilt.FILT2;
else if (tmp.ToUpper() == "FILT3") filter = MeasFilt.FILT3;
else filter = MeasFilt.NONE;

// Meas Atten
measAtten = double.Parse(line.Split(',')[16]);

// Modulation Type
//tmp = line.Split(',')[17];

// Modulation File
//tmp = line.Split(',')[18];

// Duty Cycle
//tmp = line.Split(',')[19];

// Comment
//tmp = line.Split(',')[20];

//if (!SourcePortKey.Any(x => (x.freq == SourcePortKey.Item1 && x.sigGen ==
SourcePortKey.Item2 && x.srcPort == SourcePortKey.Item3 && x.ampPort ==
SourcePortKey.Item4 && x.srcPrimary == SourcePortKey.Item5 && x.srcSecondary ==
SourcePortKey.Item6)))
if (!sourcePortCalData.Any(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen &&
x.Key.SrcPort == srcPort && x.Key.AmpPort == ampPort && x.Key.SwitchFilterPrimary ==
srcPrimary && x.Key.SwitchFilterSecondary == srcSecondary && x.Key.SwitchFilterCoupler ==
srcCoupler)))
{
SwitchFilterUserCalSourceConfig sourceKey = new SwitchFilterUserCalSourceConfig() {
Frequency = freq, SigGen = sigGen, SrcPort = srcPort, AmpPort = ampPort, SwitchFilterPrimary =
srcPrimary, SwitchFilterSecondary = srcSecondary, SwitchFilterCoupler = srcCoupler };
```

```
sourcePortCalData.Add(sourceKey, new SwitchFilterUserCalSourceData()
{
Bypass = double.Parse(line.Split(',')[21]),
A1 = double.Parse(line.Split(',')[22]),
A2 = double.Parse(line.Split(',')[23]),
A3 = double.Parse(line.Split(',')[24]),
A1_A2 = double.Parse(line.Split(',')[25]),
A1_A3 = double.Parse(line.Split(',')[26]),
A2_A3 = double.Parse(line.Split(',')[27]),
A1_A2_A3 = double.Parse(line.Split(',')[28]),
SF_Amp_Off = double.Parse(line.Split(',')[29]),
SF_Amp_On = double.Parse(line.Split(',')[30]),

CAL_Bypass = double.Parse(line.Split(',')[37]),
CAL_A1 = double.Parse(line.Split(',')[38]),
CAL_A2 = double.Parse(line.Split(',')[39]),
CAL_A3 = double.Parse(line.Split(',')[40]),
CAL_A1_A2 = double.Parse(line.Split(',')[41]),
CAL_A1_A3 = double.Parse(line.Split(',')[42]),
CAL_A2_A3 = double.Parse(line.Split(',')[43]),
CAL_A1_A2_A3 = double.Parse(line.Split(',')[44]),
});
}

//if (!MeasurePortKey.Any(x => (x.freq == MeasurePortKey.Item1 && x.measPrimary ==
MeasurePortKey.Item2 && x.measSecondary == MeasurePortKey.Item3 && x.measCoupler ==
MeasurePortKey.Item4)))
if (!measurePortCalData.Any(x => (x.Key.Frequency == freq && x.Key.MeasPort == measPort &&
x.Key.SwitchFilterPrimary == measPrimary && x.Key.SwitchFilterSecondary == measSecondary
&& x.Key.SwitchFilterCoupler == measCoupler && x.Key.MeasFilt == filter)))
{
SwitchFilterUserCalMeasureConfig measureKey = new SwitchFilterUserCalMeasureConfig() {
Frequency = freq, MeasPort = measPort, SwitchFilterPrimary = measPrimary,
SwitchFilterSecondary = measSecondary, SwitchFilterCoupler = measCoupler, MeasFilt = filter };
measurePortCalData.Add(measureKey, new SwitchFilterUserCalMeasureData()
{
Bypass = double.Parse(line.Split(',')[31]),
A1 = double.Parse(line.Split(',')[32]),
A2 = double.Parse(line.Split(',')[33]),
A1_A2 = double.Parse(line.Split(',')[34]),
SF_Amp_Off = double.Parse(line.Split(',')[35]),
SF_Amp_On = double.Parse(line.Split(',')[36]),
```

```
});
}
}
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
return -1;
}

return 0;
}

public SwitchFilterUserCalSourceData GetSrcCalFactor(double freq, SigGen sigGen, SrcPort
srcPort, CalDataRF300.AmpPort ampPort, CalDataRF300.sfPort srcPrimary,
CalDataRF300.sfPort srcSecondary, CalDataRF300.sfCouplerPort srcCoupler)
{
if (sourcePortCalData.Any(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen &&
x.Key.SrcPort == srcPort && x.Key.AmpPort == ampPort && x.Key.SwitchFilterPrimary ==
srcPrimary && x.Key.SwitchFilterSecondary == srcSecondary && x.Key.SwitchFilterCoupler ==
srcCoupler)))
{
return sourcePortCalData.First(x => (x.Key.Frequency == freq && x.Key.SigGen == sigGen &&
x.Key.SrcPort == srcPort && x.Key.AmpPort == ampPort && x.Key.SwitchFilterPrimary ==
srcPrimary && x.Key.SwitchFilterSecondary == srcSecondary && x.Key.SwitchFilterCoupler ==
srcCoupler)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

public SwitchFilterUserCalMeasureData GetMeasCalFactor(double freq, MeasPort measPort,
CalDataRF300.sfPort measPrimary, CalDataRF300.sfPort measSecondary,
CalDataRF300.sfCouplerPort measCoupler, MeasFilt filter)
{
if (measurePortCalData.Any(x => (x.Key.Frequency == freq && x.Key.MeasPort == measPort &&
```

```
x.Key.SwitchFilterPrimary == measPrimary && x.Key.SwitchFilterSecondary == measSecondary
&& x.Key.SwitchFilterCoupler == measCoupler && x.Key.MeasFilt == filter)))
{
return measurePortCalData.First(x => (x.Key.Frequency == freq && x.Key.MeasPort == measPort
&& x.Key.SwitchFilterPrimary == measPrimary && x.Key.SwitchFilterSecondary ==
measSecondary && x.Key.SwitchFilterCoupler == measCoupler && x.Key.MeasFilt ==
filter)).Value;
}
else
{
//todo: add diagnostic logging
return null;
}
}

//public bool CalFrequencyExists(double freq, SigGen sigGen, SrcPort srcPort,
CalDataRF300.AmpPort ampPort, CalDataRF300.SwitchFilter srcPrimary,
CalDataRF300.SwitchFilter srcSecondary, SrcAmpConfig srcAmpConfig)
//{
//    bool isAvailable = true;

//    try
//    {
//        double calValue = sourcePortCalData[new Tuple<double, SigGen, SrcPort,
CalDataRF300.AmpPort, CalDataRF300.SwitchFilter, CalDataRF300.SwitchFilter,
SrcAmpConfig>(freq, sigGen, srcPort, ampPort, srcPrimary, srcSecondary, srcAmpConfig)];
//    }
//    catch
//    {
//        isAvailable = false;
//    }

//    return isAvailable;
//}

//public bool CalFrequencyExists(double freq, SigGen sigGen, SrcPort srcPort,
CalDataRF300.AmpPort ampPort, CalDataRF300.SwitchFilter srcPrimary,
CalDataRF300.CouplerPort srcCoupler, bool ampEnable)
//{
//    bool isAvailable = true;

//    try
```

```
//    {
//        double calValue = sourceCouplerCalData[new Tuple<double, SigGen, SrcPort,
CalDataRF300.AmpPort, CalDataRF300.SwitchFilter, CalDataRF300.CouplerPort, bool>(freq,
sigGen, srcPort, ampPort, srcPrimary, srcCoupler, ampEnable)];
//    }
//    catch
//    {
//        isAvailable = false;
//    }

//    return isAvailable;
//}

//public bool CalFrequencyExists(double freq, CalDataRF300.SwitchFilter measPrimary,
CalDataRF300.SwitchFilter measSecondary, CalDataRF300.CouplerPort measCoupler,
MeasAmpConfig measAmpConfig)
//{
//    bool isAvailable = true;

//    try
//    {
//        double calValue = measurePortCalData[new Tuple<double, CalDataRF300.SwitchFilter,
CalDataRF300.SwitchFilter, CalDataRF300.CouplerPort, MeasAmpConfig>(freq, measPrimary,
measSecondary, measCoupler, measAmpConfig)];
//    }
//    catch
//    {
//        isAvailable = false;
//    }

//    return isAvailable;
//}

//public bool CalFrequencyExists(double freq, CalDataRF300.SwitchFilter measPrimary,
CalDataRF300.SwitchFilter measSecondary, CalDataRF300.CouplerPort measCoupler, bool
ampEnable)
//{
//    bool isAvailable = true;

//    try
//    {
//        double calValue = measureCouplerCalData[new Tuple<double, CalDataRF300.SwitchFilter,
```

```
CalDataRF300.SwitchFilter, CalDataRF300.CouplerPort, bool>(freq, measPrimary,
measSecondary, measCoupler, ampEnable)];
//    }
//    catch
//    {
//        isAvailable = false;
//    }

//    return isAvailable;
//}
}



//============================== DC Sense Calibration
==============================

public class PsmSystemCal
{
bool writeToConsole = true;

public Info info = new Info();
public Dictionary<Tuple<ushort, double, HRange>, double> dcSenseCalData = new
Dictionary<Tuple<ushort, double, HRange>, double>();
public List<double> voltageList = new List<double>();

public int ReadCalDataFile()
{
const string calDataFile = @"C:\MerlinTest\System\Calibration\PSM\PSM_DC_Cal_Data.csv";

if (writeToConsole) Console.WriteLine("Loading PSM calibration data file: {0}\n", calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "WARNING: PSM calibration data file not found:" + calDataFile + "\n";
if (writeToConsole) Console.WriteLine(message);
//MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Check if file is locked
```

```
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
}
catch (IOException ex)
{
MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Read cal data from file
ImportCalData(calDataFile);

// Convert dictionary values to array
int counter = 0;
var calDataArray = dcSenseCalData.Keys.ToArray();
do
{
voltageList.Add(calDataArray[counter].Item2);
counter += 40;
} while (counter < calDataArray.Length);

return 0;
}

public int ReadCalDataFile(string calDataFile)
{
if (writeToConsole) Console.WriteLine("Loading PSM calibration data file: {0}\n", calDataFile);

// Check if file exists
if (!File.Exists(calDataFile))
{
string message = "WARNING: PSM calibration data file not found:" + calDataFile + "\n";
if (writeToConsole) Console.WriteLine(message);
//MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}
```

```
// Check if file is locked
try
{
FileStream fs = File.Open(calDataFile, FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
fs.Close();
}
catch (IOException ex)
{
MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Read cal data from file
ImportCalData(calDataFile);

// Convert dictionary values to array
int counter = 0;
var calDataArray = dcSenseCalData.Keys.ToArray();
do
{
voltageList.Add(calDataArray[counter].Item2);
counter += 40;
} while (counter < calDataArray.Length);

return 0;
}

public int ImportCalData(string calDataFile)
{
string line = "";
string tmpString = "";
ushort source = 1;
double voltage = 0;
double calData = 0;

HRange range = HRange.R_3_5A;

Tuple<ushort, double, HRange> SenseCurrentKey = new Tuple<ushort, double, HRange>(source,
voltage, range);
```

```
try
{
using (StreamReader stream = new StreamReader(calDataFile))
{
while ((line = stream.ReadLine()) != null)
{
tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "TESTER ID:")
{
info.TesterId = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "HARDWARE REVISION:")
{
info.Revision = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "DATE:")
{
info.Date = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "COMMENT:")
{
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "VOLTAGE (V)")
{
while ((line = stream.ReadLine()) != null)
{
voltage = double.Parse(line.Split(',')[0]);

for (source = 1; source <= 4; source++)
{
for (range = HRange.R_3_5A; range <= HRange.R_5mA; range++)
{
```

```
int index = (int)(range + 1) + ((source - 1) * 10);
calData = double.Parse(line.Split(',')[(int)(range + 1) + ((source - 1) * 10)]);
SenseCurrentKey = Tuple.Create(source, voltage, range);
dcSenseCalData.Add(SenseCurrentKey, calData);
}
}
}

return 0;
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
return -1;
}

return 0;
}

public bool CalVoltageExists(double voltage)
{
bool isAvailable = true;

try
{
double calValue = dcSenseCalData[new Tuple<ushort, double, HRange>(1, voltage,
HRange.R_3_5A)];
}
catch
{
isAvailable = false;
}

return isAvailable;
}

public double GetSenseCurrentCalFactor(ushort source, double voltage, HRange range)
{
if (dcSenseCalData.Any(x => (x.Key.Item1 == source && x.Key.Item2 == voltage && x.Key.Item3
```

```
== range)))
{
return dcSenseCalData.First(x => (x.Key.Item1 == source && x.Key.Item2 == voltage &&
x.Key.Item3 == range)).Value;
}
else
{
//todo: add diagnostic logging
return double.NaN;
}
}

public double GetInterpolatedSenseCurrent(ushort source, double voltage, HRange range)
{
double lowerVoltage = 0;
double upperVoltage = 0;
double senseCurrent = double.NaN;

//long v = (long)Math.Round(voltage);
if (voltage < 0 || voltage > 8) return senseCurrent;

// Determine if voltage is below the lowest calibration voltage available
if (voltage <= voltageList[0])
{
senseCurrent = dcSenseCalData[new Tuple<ushort, double, HRange>(source, voltageList[0],
range)];
return senseCurrent;
}

// Determine if voltage is above the highest calibration voltage available
if (voltage >= voltageList[voltageList.Count - 1])
{
senseCurrent = dcSenseCalData[new Tuple<ushort, double, HRange>(source,
voltageList[voltageList.Count - 1], range)];
return senseCurrent;
}

// Find the closest calibration voltages available
for (int i = 0; i < voltageList.Count; i++)
{
if (voltageList[i] > voltage)
{
```

```
upperVoltage = voltageList[i];
lowerVoltage = voltageList[i - 1];
break;
}
}

// Get interpolated calibration value
double lowerSenseCurrent = dcSenseCalData[new Tuple<ushort, double, HRange>(source, lowerVoltage, range)];
double upperSenseCurrent = dcSenseCalData[new Tuple<ushort, double, HRange>(source, upperVoltage, range)];
double m = (upperSenseCurrent - lowerSenseCurrent) / (upperVoltage - lowerVoltage); // m = delta y / delta x
double b = lowerSenseCurrent - (m * lowerVoltage); // b = y - mx
senseCurrent = m * voltage + b; // y = mx + b

return senseCurrent;
}
}


//============= Dynamic Waveform Calibration (User Cal v5.0 and higher) ==============

public enum ModulationType { None, CW, GSM, EDGE, LTE_FDD, LTE_TDD, TDSCDMA, WCDMA, WLAN, NR };
public enum ChannelBandwidth { OnePointFourMHz, ThreeMHz, FiveMHz, TenMHz, FifteenMHz, TwentyMHz, FortyMHz, FiftyMHz, SixtyMHz, EightyMHz, OneHundredMHz, OneHundredSixtyMHz, TwoHundredMHz, NA };

public class WaveformKey
{
public string WaveformFile { get; set; }
public short Marker { get; set; }
}

public class WaveformSettings
{
// Added for dynamic waveform calibration (User Cal v5.0 and up)
public string WaveformFile { get; set; }
public ModulationType Modulation { get; set; }
public ChannelBandwidth Bandwidth { get; set; }
public short Marker { get; set; }
```

```
public double MarkerLocation { get; set; }
public double DataLocation { get; set; }
public double DataLength { get; set; }
public double MeasLength { get; set; }
public double MeasOffset { get; set; }
public double DutyCycle { get; set; }
public double FrameLength { get; set; }
public double MeasSpan { get; set; }
public double MinSampleFreq { get; set; }
public double RecSampleFreq { get; set; }
public short AnalysisSlot { get; set; }
public double Headroom { get; set; }

// Waveform marker information
public MarkerData iqsMarkerData;
}

/// <summary>
/// Class to represent the individual marker transitions.
/// </summary>
public class Transition
{
/// <summary>
/// The sample that the marker goes ON.
/// </summary>
public int OnTime { get; set; }

/// <summary>
/// The sample that the marker goes off.
/// </summary>
public int OffTime { get; set; }

/// <summary>
/// The attenuation for the marker period.
/// </summary>
public double Attenuation { get; set; }
}

/// <summary>
/// Represents the marker positions for an entire waveform.
/// </summary>
public class MarkerData
```

```
{
/// <summary>
/// Constructor for the MarkerData class.
/// </summary>
public MarkerData()
{
Marker1Info = new List<Transition>();
Marker2Info = new List<Transition>();
Marker3Info = new List<Transition>();
Marker4Info = new List<Transition>();
}

/// <summary>
/// Gets / Sets the makrer transitions for marker 1.
/// </summary>
public List<Transition> Marker1Info { get; set; }

/// <summary>
/// Gets / Sets the makrer transitions for marker 2.
/// </summary>
public List<Transition> Marker2Info { get; set; }

/// <summary>
/// Gets / Sets the makrer transitions for marker 3.
/// </summary>
public List<Transition> Marker3Info { get; set; }

/// <summary>
/// Gets / Sets the makrer transitions for marker 4.
/// </summary>
public List<Transition> Marker4Info { get; set; }
}

public class GsmWaveformInfo
{
public MarkerData Marker { get; set; }
public bool[] SlotEnabled { get; set; }
public double SymbolRate { get; set; }
}

public class LteWaveformInfo
{
```

```
public MarkerData Marker { get; set; }
public bool[] SlotEnabled { get; set; }
public ChannelBandwidth Bandwidth { get; set; }
}

public class TdscdmaWaveformInfo
{
public MarkerData Marker { get; set; }
public bool[] SlotEnabled { get; set; }
public bool UpPtsEnabled { get; set; }
public bool DwPtsEnabled { get; set; }
}

public class WcdmaWaveformInfo
{
public MarkerData Marker { get; set; }
public bool[] SlotEnabled { get; set; }
}

public class WlanWaveformInfo
{
public MarkerData Marker { get; set; }
public double IdleTime { get; set; }
public ChannelBandwidth Bandwidth { get; set; }
}

public class NrWaveformInfo
{
public MarkerData Marker { get; set; }
public bool[] SlotEnabled { get; set; }
public ChannelBandwidth Bandwidth { get; set; }
}

public class Waveform
{
//public int ReadWaveformFile(ModulationType modType)
//{
//    string waveformFile = "";

//    if (modType == ModulationType.None)
//    {
//    }
```

```
//    else if (modType == ModulationType.CW)
//    {
//        waveformFile = "CW_Waveform_List.csv";
//    }
//    else if (modType == ModulationType.GSM)
//    {
//        waveformFile = "GSM_Waveform_List.csv";
//    }
//    else if (modType == ModulationType.EDGE)
//    {
//        waveformFile = "EDGE_Waveform_List.csv";
//    }
//    else if (modType == ModulationType.LTE_FDD || modType == ModulationType.LTE_TDD)
//    {
//        waveformFile = "LTE_Waveform_List.csv";
//    }
//    else if (modType == ModulationType.TDSCDMA)
//    {
//        waveformFile = "TDSCDMA_Waveform_List.csv";
//    }
//    else if (modType == ModulationType.WCDMA)
//    {
//        waveformFile = "WCDMA_Waveform_List.csv";
//    }
//    else if (modType == ModulationType.WLAN)
//    {
//        waveformFile = "WLAN_Waveform_List.csv";
//    }

//    string waveformTarget = Path.Combine(@"C:\MerlinTest\Waveform Library\", waveformFile);

//    if (writeToConsole) Console.WriteLine("Loading modulation waveform file: {0}\n",
waveformTarget);

//    // Check if file exists
//    if (!File.Exists(waveformTarget))
//    {
//        string message = "ERROR: Modulation waveform list not found:" + waveformTarget + "\n";
//        if (writeToConsole) Console.WriteLine(message);
//        //MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
//        return -1;
```

```
//    }

//    // Check if file is locked
//    bool fileLocked = true;
//    do
//    {
//        try
//        {
//            FileStream fs = File.Open(waveformTarget, FileMode.OpenOrCreate,
FileAccess.ReadWrite, FileShare.None);
//            fs.Close();
//            fileLocked = false;
//        }
//        catch (IOException ex)
//        {
//            MessageBox.Show(ex.Message + "\n\nPlease close file to continue...", "Merlin Test
Technologies", MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1);
//        }
//    } while (fileLocked == true);

//    // Read cal data from file
//    ImportWaveformData(waveformTarget);

//    return 0;
//}

//public int ImportWaveformData(string waveformFile)
//{
//    string line = "";

//    //Tuple<double, SrcAmpConfig> AmpGainKey = new Tuple<double, SrcAmpConfig>(freq,
srcAmpConfig);
//    //Tuple<double, SrcPort, SrcAmpConfig> SourcePortKey = new Tuple<double, SrcPort,
SrcAmpConfig>(freq, srcPort, srcAmpConfig);
//    //Tuple<double, MeasPort, SrcMeasAmpConfig> MeasurePortKey = new Tuple<double,
MeasPort, SrcMeasAmpConfig>(freq, measPort, measAmpConfig);

//    // Check if file exists
//    //if (!File.Exists(waveformFile))
//    //{
//    //    string message = "ERROR: Waveform configuration file not found!\n" + waveformFile;
```

```
//    //    if (writeToConsole) Console.WriteLine(message);
//    //    //MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
//    //    return -1;
//    //}

//    // Check if file is locked
//    //bool fileLocked = true;
//    //do
//    //{
//    //    try
//    //    {
//    //        FileStream fs = File.Open(waveformFile, FileMode.OpenOrCreate,
FileAccess.ReadWrite, FileShare.None);
//    //        fs.Close();
//    //        fileLocked = false;
//    //    }
//    //    catch (IOException ex)
//    //    {
//    //        MessageBox.Show(ex.Message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
//    //    }
//    //} while (fileLocked == true);

//    // Read waveform file contents
//    try
//    {
//        using (StreamReader stream = new StreamReader(waveformFile))
//        {
//            while ((line = stream.ReadLine()) != null)
//            {
//                string tmpString = line.Split(',')[0];

//                if (tmpString.ToUpper() == "PRODUCT:")
//                {
//                    info.Product = line.Split(',')[1];
//                }
//                else if (tmpString.ToUpper() == "VERSION:")
//                {
//                    info.CalVersion = line.Split(',')[1];
//                }
//                else if (tmpString.ToUpper() == "COMMENT:")
```

```
//             {
//                 info.Comment = line.Split(',')[1];
//             }
//             else if (tmpString.ToUpper() == "WAVEFORM FILE:")
//             {
//                 while ((line = stream.ReadLine()) != "")
//                 {
//                     Configuration waveformParams = new Configuration();

//                     waveformParams.Modulation = ModulationType.LTE_FDD;
//                     waveformParams.WaveformFile = line.Split(',')[0];

//                     double bandwidth = double.Parse(line.Split(',')[1]);
//                     if (bandwidth == 1.4) waveformParams.Bandwidth =
ChannelBandwidth.OnePointFourMHz;
//                     else if (bandwidth == 3) waveformParams.Bandwidth =
ChannelBandwidth.ThreeMHz;
//                     else if (bandwidth == 5) waveformParams.Bandwidth =
ChannelBandwidth.FiveMHz;
//                     else if (bandwidth == 10) waveformParams.Bandwidth =
ChannelBandwidth.TenMHz;
//                     else if (bandwidth == 15) waveformParams.Bandwidth =
ChannelBandwidth.FifteenMHz;
//                     else if (bandwidth == 20) waveformParams.Bandwidth =
ChannelBandwidth.TwentyMHz;

//                     waveformParams.Marker = ushort.Parse(line.Split(',')[2]);
//                     waveformParams.MarkerLocation = double.Parse(line.Split(',')[3]);
//                     waveformParams.DataLocation = double.Parse(line.Split(',')[4]);
//                     waveformParams.DataLength = double.Parse(line.Split(',')[5]);
//                     waveformParams.DutyCycle = double.Parse(line.Split(',')[6]);

//                     setup.Add(waveformParams);
//                 }
//             }
//         }
//     }
//     catch (FileNotFoundException error)
//     {
//         Console.WriteLine("\n{0}", error.Message);
//         return -1;
```

```
//    }

//    return 0;
//}

public GsmWaveformInfo GetGsmEdgeWaveformInfo(string FileName)
{
GsmWaveformInfo info = new GsmWaveformInfo();

// Loading from a file, you can also load from a stream
var xml = XDocument.Load(FileName);

// -------------------- Get waveform markers --------------------

MarkerData mdata = new MarkerData();

// Query the data and find the section containing the marker information
var query = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Transitions"
select c;

// As the .IQS fileformt is not very XML friendly I need to skip the
// first entry that has the attribute Transitions. This is done by turning
// the query into a list and the accessing the different marker positions as an
// array entry.
var myList = query.ToList();

#region Marker 1

// Now Find the first Transition(s)
var Marker1Cont = from c in myList[1].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker1Ontime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker1Offtime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
```

```
select c).ToList();

// Now Get Attenuation.
var Marker1Atten = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert elements to correct datatype
for (int index = 0; index < Marker1Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker1Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an integer.", Marker1Ontime[index].Value));
}

if (int.TryParse(Marker1Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an integer.", Marker1Offtime[index].Value));
}

if (double.TryParse(Marker1Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker1Atten[index].Value));
}

mdata.Marker1Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 1

#region Marker 2

var Marker2Cont = from c in myList[2].Descendants("section")
where (string)c.Attribute("name") == "Transition"
```

```
select c;

// Now get ON Time.
var Marker2Ontime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker2Offtime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker2Atten = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker2Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker2Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an
integer.", Marker2Ontime[index].Value));
}

if (int.TryParse(Marker2Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an
integer.", Marker2Offtime[index].Value));
}

if (double.TryParse(Marker2Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker2Atten[index].Value));
}
```

```
mdata.Marker2Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 2

#region Marker 3

// Now Find the first Transition(s)
var Marker3Cont = from c in myList[3].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker3Ontime = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker3Offtime = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker3Atten = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker3Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker3Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 3 On Time Value of {0} couldn't be converted to an
integer.", Marker3Ontime[index].Value));
}

if (int.TryParse(Marker3Offtime[index].Value, out offTime) == false)
```

```
{
throw new Exception(string.Format("Marker 3 Off Time Value of {0} couldn't be converted to an integer.", Marker3Offtime[index].Value));
}

if (double.TryParse(Marker3Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 3 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker3Atten[index].Value));
}

mdata.Marker3Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 3

#region Marker 4

// Now Find the Marker 1 section.
//var marker4Section = from c in query.First().Descendants("section")
//                     where (string)c.Attribute("name") == "3"
//                     select c;

var Marker4Cont = from c in myList[4].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker4Ontime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker4Offtime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker4Atten = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();
```

```
// Convert Elements to  correct datatype.
for (int index = 0; index < Marker4Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker4Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 4 On Time Value of {0} couldn't be converted to an
integer.", Marker4Ontime[index].Value));
}

if (int.TryParse(Marker4Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 4 Off Time Value of {0} couldn't be converted to an
integer.", Marker4Offtime[index].Value));
}

if (double.TryParse(Marker4Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 4 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker4Atten[index].Value));
}


mdata.Marker4Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 4

info.Marker = mdata;

// -------------------- Get enabled slots --------------------

info.SlotEnabled = new bool[8];

var querySlotState= from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Slot"
select c;
```

```
var listSlotState = querySlotState.ToList();

// Get slot enable state
var slotState = (from c in listSlotState[0].Descendants("key")
where (string)c.Attribute("name") == "SlotState"
select c).ToList();

// Convert elements to correct datatype
for (int index = 0; index < slotState.Count; index++)
{
bool slotEnabled = false;

if (bool.TryParse(slotState[index].Value, out slotEnabled) == false)
{
throw new Exception(string.Format("Slot enabled state of {0} couldn't be converted to a bool.",
slotState[index].Value));
}

info.SlotEnabled[index] = slotEnabled;
}

// ---------- Get symbol rate ----------

var querySymbolRate = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "GSM Generation"
select c;

var listSymbolRate = querySymbolRate.ToList();

// Get symbol rate
var symbolRate = (from c in listSymbolRate[0].Descendants("key")
where (string)c.Attribute("name") == "SymbolRate"
select c).ToList();

// Convert elements to correct datatype
if (double.TryParse(symbolRate[0].Value, out double rate) == false)
{
throw new Exception(string.Format("Slot rate of {0} couldn't be converted to a double.",
symbolRate[0].Value));
}
```

```
info.SymbolRate = rate;

return info;
}

public LteWaveformInfo GetLteWaveformInfo(string FileName, ModulationType modulation)
{
LteWaveformInfo info = new LteWaveformInfo();

// Loading from a file, you can also load from a stream
var xml = XDocument.Load(FileName);

// -------------------- Get waveform markers --------------------

MarkerData mdata = new MarkerData();

// Query the data and find the section containing the marker information
var query = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Transitions"
select c;

// As the .IQS fileformt is not very XML friendly I need to skip the
// first entry that has the attribute Transitions. This is done by turning
// the query into a list and the accessing the different marker positions as an
// array entry.
var myList = query.ToList();

#region Marker 1

// Now Find the first Transition(s)
var Marker1Cont = from c in myList[1].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker1Ontime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker1Offtime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
```

```
select c).ToList();

// Now Get Attenuation.
var Marker1Atten = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert elements to correct datatype
for (int index = 0; index < Marker1Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker1Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an integer.", Marker1Ontime[index].Value));
}

if (int.TryParse(Marker1Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an integer.", Marker1Offtime[index].Value));
}

if (double.TryParse(Marker1Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker1Atten[index].Value));
}

mdata.Marker1Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 1

#region Marker 2

var Marker2Cont = from c in myList[2].Descendants("section")
where (string)c.Attribute("name") == "Transition"
```

```
select c;

// Now get ON Time.
var Marker2Ontime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker2Offtime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker2Atten = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker2Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker2Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an
integer.", Marker2Ontime[index].Value));
}

if (int.TryParse(Marker2Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an
integer.", Marker2Offtime[index].Value));
}

if (double.TryParse(Marker2Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker2Atten[index].Value));
}
```

```
mdata.Marker2Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 2

#region Marker 3

// Now Find the first Transition(s)
var Marker3Cont = from c in myList[3].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker3Ontime = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker3Offtime = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker3Atten = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker3Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker3Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 3 On Time Value of {0} couldn't be converted to an
integer.", Marker3Ontime[index].Value));
}

if (int.TryParse(Marker3Offtime[index].Value, out offTime) == false)
```

```
{
throw new Exception(string.Format("Marker 3 Off Time Value of {0} couldn't be converted to an integer.", Marker3Offtime[index].Value));
}

if (double.TryParse(Marker3Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 3 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker3Atten[index].Value));
}

mdata.Marker3Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 3

#region Marker 4

// Now Find the Marker 1 section.
//var marker4Section = from c in query.First().Descendants("section")
//                     where (string)c.Attribute("name") == "3"
//                     select c;

var Marker4Cont = from c in myList[4].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker4Ontime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker4Offtime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker4Atten = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();
```

```
// Convert Elements to  correct datatype.
for (int index = 0; index < Marker4Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker4Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 4 On Time Value of {0} couldn't be converted to an
integer.", Marker4Ontime[index].Value));
}

if (int.TryParse(Marker4Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 4 Off Time Value of {0} couldn't be converted to an
integer.", Marker4Offtime[index].Value));
}

if (double.TryParse(Marker4Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 4 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker4Atten[index].Value));
}


mdata.Marker4Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 4

info.Marker = mdata;

// -------------------- Get enabled slots --------------------

info.SlotEnabled = new bool[20];

if (modulation == ModulationType.LTE_FDD)
{
for (int slotIndex = 0; slotIndex < info.SlotEnabled.Length; slotIndex++)
```

```
{
info.SlotEnabled[slotIndex] = true;
}
}
else if (modulation == ModulationType.LTE_TDD)
{
var queryConfiguration = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "CarrierDefinition"
select c;

var listConfiguration = queryConfiguration.ToList();

// Get slot enable state
var configuration = (from c in listConfiguration[0].Descendants("key")
where (string)c.Attribute("name") == "UplinkDownlinkConfiguration"
select c).ToList();

// Convert elements to correct datatype
if (int.TryParse(configuration[0].Value, out int config) == false)
{
throw new Exception(string.Format("Slot configuration {0} couldn't be converted to an integer.",
configuration[0].Value));
}

if (config == 0)
{
info.SlotEnabled[0] = false;
info.SlotEnabled[1] = false;
info.SlotEnabled[2] = false;
info.SlotEnabled[3] = false;
info.SlotEnabled[4] = true;
info.SlotEnabled[5] = true;
info.SlotEnabled[6] = true;
info.SlotEnabled[7] = true;
info.SlotEnabled[8] = true;
info.SlotEnabled[9] = true;
info.SlotEnabled[10] = false;
info.SlotEnabled[11] = false;
info.SlotEnabled[12] = false;
info.SlotEnabled[13] = false;
info.SlotEnabled[14] = true;
info.SlotEnabled[15] = true;
```

```
info.SlotEnabled[16] = true;
info.SlotEnabled[17] = true;
info.SlotEnabled[18] = true;
info.SlotEnabled[19] = true;
}
else if (config == 1)
{
info.SlotEnabled[0] = false;
info.SlotEnabled[1] = false;
info.SlotEnabled[2] = false;
info.SlotEnabled[3] = false;
info.SlotEnabled[4] = true;
info.SlotEnabled[5] = true;
info.SlotEnabled[6] = true;
info.SlotEnabled[7] = true;
info.SlotEnabled[8] = false;
info.SlotEnabled[9] = false;
info.SlotEnabled[10] = false;
info.SlotEnabled[11] = false;
info.SlotEnabled[12] = false;
info.SlotEnabled[13] = false;
info.SlotEnabled[14] = true;
info.SlotEnabled[15] = true;
info.SlotEnabled[16] = true;
info.SlotEnabled[17] = true;
info.SlotEnabled[18] = false;
info.SlotEnabled[19] = false;
}
else if (config == 2)
{
info.SlotEnabled[0] = false;
info.SlotEnabled[1] = false;
info.SlotEnabled[2] = false;
info.SlotEnabled[3] = false;
info.SlotEnabled[4] = true;
info.SlotEnabled[5] = true;
info.SlotEnabled[6] = false;
info.SlotEnabled[7] = false;
info.SlotEnabled[8] = false;
info.SlotEnabled[9] = false;
info.SlotEnabled[10] = false;
info.SlotEnabled[11] = false;
```

```
info.SlotEnabled[12] = false;
info.SlotEnabled[13] = false;
info.SlotEnabled[14] = true;
info.SlotEnabled[15] = true;
info.SlotEnabled[16] = false;
info.SlotEnabled[17] = false;
info.SlotEnabled[18] = false;
info.SlotEnabled[19] = false;
}
else if (config == 3)
{
info.SlotEnabled[0] = false;
info.SlotEnabled[1] = false;
info.SlotEnabled[2] = false;
info.SlotEnabled[3] = false;
info.SlotEnabled[4] = true;
info.SlotEnabled[5] = true;
info.SlotEnabled[6] = true;
info.SlotEnabled[7] = true;
info.SlotEnabled[8] = true;
info.SlotEnabled[9] = true;
info.SlotEnabled[10] = false;
info.SlotEnabled[11] = false;
info.SlotEnabled[12] = false;
info.SlotEnabled[13] = false;
info.SlotEnabled[14] = false;
info.SlotEnabled[15] = false;
info.SlotEnabled[16] = false;
info.SlotEnabled[17] = false;
info.SlotEnabled[18] = false;
info.SlotEnabled[19] = false;
}
else if (config == 4)
{
info.SlotEnabled[0] = false;
info.SlotEnabled[1] = false;
info.SlotEnabled[2] = false;
info.SlotEnabled[3] = false;
info.SlotEnabled[4] = true;
info.SlotEnabled[5] = true;
info.SlotEnabled[6] = true;
info.SlotEnabled[7] = true;
```

```
info.SlotEnabled[8] = false;
info.SlotEnabled[9] = false;
info.SlotEnabled[10] = false;
info.SlotEnabled[11] = false;
info.SlotEnabled[12] = false;
info.SlotEnabled[13] = false;
info.SlotEnabled[14] = false;
info.SlotEnabled[15] = false;
info.SlotEnabled[16] = false;
info.SlotEnabled[17] = false;
info.SlotEnabled[18] = false;
info.SlotEnabled[19] = false;
}
else if (config == 5)
{
info.SlotEnabled[0] = false;
info.SlotEnabled[1] = false;
info.SlotEnabled[2] = false;
info.SlotEnabled[3] = false;
info.SlotEnabled[4] = true;
info.SlotEnabled[5] = true;
info.SlotEnabled[6] = false;
info.SlotEnabled[7] = false;
info.SlotEnabled[8] = false;
info.SlotEnabled[9] = false;
info.SlotEnabled[10] = false;
info.SlotEnabled[11] = false;
info.SlotEnabled[12] = false;
info.SlotEnabled[13] = false;
info.SlotEnabled[14] = false;
info.SlotEnabled[15] = false;
info.SlotEnabled[16] = false;
info.SlotEnabled[17] = false;
info.SlotEnabled[18] = false;
info.SlotEnabled[19] = false;
}
else if (config == 6)
{
info.SlotEnabled[0] = false;
info.SlotEnabled[1] = false;
info.SlotEnabled[2] = false;
info.SlotEnabled[3] = false;
```

```
info.SlotEnabled[4] = true;
info.SlotEnabled[5] = true;
info.SlotEnabled[6] = true;
info.SlotEnabled[7] = true;
info.SlotEnabled[8] = true;
info.SlotEnabled[9] = true;
info.SlotEnabled[10] = false;
info.SlotEnabled[11] = false;
info.SlotEnabled[12] = false;
info.SlotEnabled[13] = false;
info.SlotEnabled[14] = true;
info.SlotEnabled[15] = true;
info.SlotEnabled[16] = true;
info.SlotEnabled[17] = true;
info.SlotEnabled[18] = false;
info.SlotEnabled[19] = false;
}
}

// ---------- Get bandwidth ----------

var queryBandwidth = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "CarrierDefinitionManager"
select c;

var listBandwidth = queryBandwidth.ToList();

// Get bandwidth
var bandwidth = (from c in listBandwidth[0].Descendants("key")
where (string)c.Attribute("name") == "Bandwidth"
select c).ToList();

// Convert elements to correct datatype
if (bandwidth[0].Value.Contains("1p4MHz")) info.Bandwidth =
ChannelBandwidth.OnePointFourMHz;
else if (bandwidth[0].Value.Contains("bw3MHz")) info.Bandwidth = ChannelBandwidth.ThreeMHz;
else if (bandwidth[0].Value.Contains("5MHz")) info.Bandwidth = ChannelBandwidth.FiveMHz;
else if (bandwidth[0].Value.Contains("10MHz")) info.Bandwidth = ChannelBandwidth.TenMHz;
else if (bandwidth[0].Value.Contains("15MHz")) info.Bandwidth = ChannelBandwidth.FifteenMHz;
else if (bandwidth[0].Value.Contains("20MHz")) info.Bandwidth = ChannelBandwidth.TwentyMHz;
else
{
```

```
throw new Exception(string.Format("LTE bandwidth {0} is not valid.", bandwidth[0].Value));
}

return info;
}

public TdscdmaWaveformInfo GetTdscdmaWaveformInfo(string FileName)
{
TdscdmaWaveformInfo info = new TdscdmaWaveformInfo();

// Loading from a file, you can also load from a stream
var xml = XDocument.Load(FileName);

// -------------------- Get waveform markers --------------------

MarkerData mdata = new MarkerData();

// Query the data and find the section containing the marker information
var query = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Transitions"
select c;

// As the .IQS fileformt is not very XML friendly I need to skip the
// first entry that has the attribute Transitions. This is done by turning
// the query into a list and the accessing the different marker positions as an
// array entry.
var myList = query.ToList();

#region Marker 1

// Now Find the first Transition(s)
var Marker1Cont = from c in myList[1].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker1Ontime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker1Offtime = (from c in Marker1Cont.Descendants("key")
```

```
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker1Atten = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert elements to correct datatype
for (int index = 0; index < Marker1Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker1Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an
integer.", Marker1Ontime[index].Value));
}

if (int.TryParse(Marker1Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an
integer.", Marker1Offtime[index].Value));
}

if (double.TryParse(Marker1Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker1Atten[index].Value));
}

mdata.Marker1Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 1

#region Marker 2

var Marker2Cont = from c in myList[2].Descendants("section")
```

```
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker2Ontime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker2Offtime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker2Atten = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker2Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker2Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an
integer.", Marker2Ontime[index].Value));
}

if (int.TryParse(Marker2Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an
integer.", Marker2Offtime[index].Value));
}

if (double.TryParse(Marker2Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker2Atten[index].Value));
}
```

```
mdata.Marker2Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 2

#region Marker 3

// Now Find the first Transition(s)
var Marker3Cont = from c in myList[3].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker3Ontime = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker3Offtime = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker3Atten = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker3Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker3Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 3 On Time Value of {0} couldn't be converted to an
integer.", Marker3Ontime[index].Value));
}
```

```
if (int.TryParse(Marker3Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 3 Off Time Value of {0} couldn't be converted to an integer.", Marker3Offtime[index].Value));
}

if (double.TryParse(Marker3Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 3 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker3Atten[index].Value));
}

mdata.Marker3Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 3

#region Marker 4

// Now Find the Marker 1 section.
//var marker4Section = from c in query.First().Descendants("section")
//                     where (string)c.Attribute("name") == "3"
//                     select c;

var Marker4Cont = from c in myList[4].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker4Ontime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker4Offtime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker4Atten = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
```

```
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker4Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker4Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 4 On Time Value of {0} couldn't be converted to an
integer.", Marker4Ontime[index].Value));
}

if (int.TryParse(Marker4Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 4 Off Time Value of {0} couldn't be converted to an
integer.", Marker4Offtime[index].Value));
}

if (double.TryParse(Marker4Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 4 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker4Atten[index].Value));
}


mdata.Marker4Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 4

info.Marker = mdata;

// -------------------- Get enabled slots --------------------

info.SlotEnabled = new bool[7];

var querySlotState = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Base Traffic Slot Engine"
```

```
select c;

var listSlotState = querySlotState.ToList();

// Get slot enable state
var slotState = (from c in listSlotState[0].Descendants("key")
where (string)c.Attribute("name") == "TrafficSlotState"
select c).ToList();

// Convert elements to correct datatype
for (int index = 0; index < slotState.Count; index++)
{
bool slotEnabled = false;

if (bool.TryParse(slotState[index].Value, out slotEnabled) == false)
{
throw new Exception(string.Format("Slot enabled state of {0} couldn't be converted to a bool.",
slotState[index].Value));
}

info.SlotEnabled[index] = slotEnabled;
}

// ---------- Get UpPTS slot state ----------

info.UpPtsEnabled = false;

var queryUpPtsState = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "TD-SCDMA Generation"
select c;

var listUpPtsState = queryUpPtsState.ToList();

// Get slot enable state
var UpPtsState = (from c in listUpPtsState[0].Descendants("key")
where (string)c.Attribute("name") == "UpPtsState"
select c).ToList();

// Convert elements to correct datatype
if (bool.TryParse(UpPtsState[0].Value, out bool UpPtsEnabled) == false)
{
throw new Exception(string.Format("Slot enabled state of {0} couldn't be converted to a bool.",
```

```
UpPtsState[0].Value));
}

info.UpPtsEnabled = UpPtsEnabled;

// -------------------- Get DownPTS slot state --------------------

info.DwPtsEnabled = false;

var queryDwPtsState = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "TD-SCDMA Generation"
select c;

var listDwPtsState = queryDwPtsState.ToList();

// Get slot enable state
var DwPtsState = (from c in listDwPtsState[0].Descendants("key")
where (string)c.Attribute("name") == "DwPtsState"
select c).ToList();

// Convert elements to correct datatype
if (bool.TryParse(DwPtsState[0].Value, out bool DwPtsEnabled) == false)
{
throw new Exception(string.Format("Slot enabled state of {0} couldn't be converted to a bool.",
DwPtsState[0].Value));
}

info.DwPtsEnabled = DwPtsEnabled;

return info;
}

public WcdmaWaveformInfo GetWcdmaWaveformInfo(string FileName)
{
WcdmaWaveformInfo info = new WcdmaWaveformInfo();

// Loading from a file, you can also load from a stream
var xml = XDocument.Load(FileName);

// -------------------- Get waveform markers --------------------

MarkerData mdata = new MarkerData();
```

```
// Query the data and find the section containing the marker information
var query = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Transitions"
select c;

// As the .IQS fileformt is not very XML friendly I need to skip the
// first entry that has the attribute Transitions. This is done by turning
// the query into a list and the accessing the different marker positions as an
// array entry.
var myList = query.ToList();

#region Marker 1

// Now Find the first Transition(s)
var Marker1Cont = from c in myList[1].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker1Ontime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker1Offtime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker1Atten = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert elements to correct datatype
for (int index = 0; index < Marker1Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker1Ontime[index].Value, out onTime) == false)
```

```
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an integer.", Marker1Ontime[index].Value));
}

if (int.TryParse(Marker1Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an integer.", Marker1Offtime[index].Value));
}

if (double.TryParse(Marker1Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker1Atten[index].Value));
}

mdata.Marker1Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 1

#region Marker 2

var Marker2Cont = from c in myList[2].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker2Ontime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker2Offtime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker2Atten = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
```

```
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker2Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker2Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an
integer.", Marker2Ontime[index].Value));
}

if (int.TryParse(Marker2Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an
integer.", Marker2Offtime[index].Value));
}

if (double.TryParse(Marker2Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker2Atten[index].Value));
}

mdata.Marker2Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 2

#region Marker 3

// Now Find the first Transition(s)
var Marker3Cont = from c in myList[3].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker3Ontime = (from c in Marker3Cont.Descendants("key")
```

```
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker3Offtime = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker3Atten = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker3Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker3Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 3 On Time Value of {0} couldn't be converted to an
integer.", Marker3Ontime[index].Value));
}

if (int.TryParse(Marker3Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 3 Off Time Value of {0} couldn't be converted to an
integer.", Marker3Offtime[index].Value));
}

if (double.TryParse(Marker3Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 3 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker3Atten[index].Value));
}

mdata.Marker3Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}
```

```
#endregion Marker 3

#region Marker 4

// Now Find the Marker 1 section.
//var marker4Section = from c in query.First().Descendants("section")
//                     where (string)c.Attribute("name") == "3"
//                     select c;

var Marker4Cont = from c in myList[4].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker4Ontime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker4Offtime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker4Atten = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker4Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker4Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 4 On Time Value of {0} couldn't be converted to an
integer.", Marker4Ontime[index].Value));
}

if (int.TryParse(Marker4Offtime[index].Value, out offTime) == false)
```

```
{
throw new Exception(string.Format("Marker 4 Off Time Value of {0} couldn't be converted to an integer.", Marker4Offtime[index].Value));
}

if (double.TryParse(Marker4Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 4 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker4Atten[index].Value));
}


mdata.Marker4Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 4

info.Marker = mdata;

// -------------------- Get enabled slots --------------------

info.SlotEnabled = new bool[20];

for (int slotIndex = 0; slotIndex < info.SlotEnabled.Length; slotIndex++)
{
info.SlotEnabled[slotIndex] = true;
}

return info;
}

public WlanWaveformInfo GetWlanWaveformInfo(string FileName)
{
WlanWaveformInfo info = new WlanWaveformInfo();

// Loading from a file, you can also load from a stream
var xml = XDocument.Load(FileName);

// -------------------- Get waveform markers --------------------

MarkerData mdata = new MarkerData();
```

```
// Query the data and find the section containing the marker information
var query = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Transitions"
select c;

// As the .IQS fileformt is not very XML friendly I need to skip the
// first entry that has the attribute Transitions. This is done by turning
// the query into a list and the accessing the different marker positions as an
// array entry.
var myList = query.ToList();

#region Marker 1

// Now Find the first Transition(s)
var Marker1Cont = from c in myList[1].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker1Ontime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker1Offtime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker1Atten = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert elements to correct datatype
for (int index = 0; index < Marker1Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker1Ontime[index].Value, out onTime) == false)
```

```
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an integer.", Marker1Ontime[index].Value));
}

if (int.TryParse(Marker1Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an integer.", Marker1Offtime[index].Value));
}

if (double.TryParse(Marker1Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker1Atten[index].Value));
}

mdata.Marker1Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 1

#region Marker 2

var Marker2Cont = from c in myList[2].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker2Ontime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker2Offtime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker2Atten = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
```

```
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker2Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker2Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an
integer.", Marker2Ontime[index].Value));
}

if (int.TryParse(Marker2Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an
integer.", Marker2Offtime[index].Value));
}

if (double.TryParse(Marker2Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker2Atten[index].Value));
}

mdata.Marker2Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 2

#region Marker 3

// Now Find the first Transition(s)
var Marker3Cont = from c in myList[3].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker3Ontime = (from c in Marker3Cont.Descendants("key")
```

```
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker3Offtime = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker3Atten = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker3Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker3Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 3 On Time Value of {0} couldn't be converted to an
integer.", Marker3Ontime[index].Value));
}

if (int.TryParse(Marker3Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 3 Off Time Value of {0} couldn't be converted to an
integer.", Marker3Offtime[index].Value));
}

if (double.TryParse(Marker3Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 3 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker3Atten[index].Value));
}

mdata.Marker3Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}
```

```
#endregion Marker 3

#region Marker 4

// Now Find the Marker 1 section.
//var marker4Section = from c in query.First().Descendants("section")
//                     where (string)c.Attribute("name") == "3"
//                     select c;

var Marker4Cont = from c in myList[4].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker4Ontime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker4Offtime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker4Atten = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker4Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker4Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 4 On Time Value of {0} couldn't be converted to an
integer.", Marker4Ontime[index].Value));
}

if (int.TryParse(Marker4Offtime[index].Value, out offTime) == false)
```

```
{
throw new Exception(string.Format("Marker 4 Off Time Value of {0} couldn't be converted to an integer.", Marker4Offtime[index].Value));
}

if (double.TryParse(Marker4Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 4 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker4Atten[index].Value));
}


mdata.Marker4Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 4

info.Marker = mdata;

// -------------------- Get idle time --------------------

var queryIdleTime = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Wireless LAN Generation"
select c;

var listIdleTime = queryIdleTime.ToList();

// Get slot enable state
var idleTime = (from c in listIdleTime[0].Descendants("key")
where (string)c.Attribute("name") == "IdleTime"
select c).ToList();

// Convert elements to correct datatype
if (double.TryParse(idleTime[0].Value, out double time) == false)
{
throw new Exception(string.Format("Idle time {0} couldn't be converted to adouble.", idleTime[0].Value));
}

info.IdleTime = time / 1e6;
```

```
// -------------------- Get bandwidth --------------------

var queryBandwidth = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Wireless LAN Generation"
select c;

var listBandwidth = queryBandwidth.ToList();

// Get bandwidth
var bandwidth = (from c in listBandwidth[0].Descendants("key")
where (string)c.Attribute("name") == "ChannelBandWidth"
select c).ToList();

// Convert elements to correct datatype
if (bandwidth[0].Value.Contains("chbw5")) info.Bandwidth = ChannelBandwidth.FiveMHz;
else if (bandwidth[0].Value.Contains("chbw10")) info.Bandwidth = ChannelBandwidth.TenMHz;
else if (bandwidth[0].Value.Contains("chbw20")) info.Bandwidth = ChannelBandwidth.TwentyMHz;
else if (bandwidth[0].Value.Contains("chbw40")) info.Bandwidth = ChannelBandwidth.FortyMHz;
else if (bandwidth[0].Value.Contains("chbw80")) info.Bandwidth = ChannelBandwidth.EightyMHz;
else if (bandwidth[0].Value.Contains("chbw160")) info.Bandwidth =
ChannelBandwidth.OneHundredSixtyMHz;
else
{
throw new Exception(string.Format("WLAN channel bandwidth {0} is not valid.",
bandwidth[0].Value));
}

return info;
}

public NrWaveformInfo GetNrWaveformInfo(string FileName)
{
NrWaveformInfo info = new NrWaveformInfo();

// Loading from a file, you can also load from a stream
var xml = XDocument.Load(FileName);

#if false
// -------------------- Get waveform markers --------------------

MarkerData mdata = new MarkerData();
```

```
// Query the data and find the section containing the marker information
var query = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Transitions"
select c;

// As the .IQS fileformt is not very XML friendly I need to skip the
// first entry that has the attribute Transitions. This is done by turning
// the query into a list and the accessing the different marker positions as an
// array entry.
var myList = query.ToList();

#region Marker 1

// Now Find the first Transition(s)
var Marker1Cont = from c in myList[1].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker1Ontime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker1Offtime = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker1Atten = (from c in Marker1Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert elements to correct datatype
for (int index = 0; index < Marker1Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker1Ontime[index].Value, out onTime) == false)
{
```

```
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an integer.", Marker1Ontime[index].Value));
}

if (int.TryParse(Marker1Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an integer.", Marker1Offtime[index].Value));
}

if (double.TryParse(Marker1Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker1Atten[index].Value));
}

mdata.Marker1Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 1

#region Marker 2

var Marker2Cont = from c in myList[2].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker2Ontime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker2Offtime = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker2Atten = (from c in Marker2Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();
```

```
// Convert Elements to  correct datatype.
for (int index = 0; index < Marker2Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker2Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 1 On Time Value of {0} couldn't be converted to an
integer.", Marker2Ontime[index].Value));
}

if (int.TryParse(Marker2Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 1 Off Time Value of {0} couldn't be converted to an
integer.", Marker2Offtime[index].Value));
}

if (double.TryParse(Marker2Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 1 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker2Atten[index].Value));
}

mdata.Marker2Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 2

#region Marker 3

// Now Find the first Transition(s)
var Marker3Cont = from c in myList[3].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker3Ontime = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
```

```
select c).ToList();

// Now get Off Time
var Marker3Offtime = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker3Atten = (from c in Marker3Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker3Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker3Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 3 On Time Value of {0} couldn't be converted to an
integer.", Marker3Ontime[index].Value));
}

if (int.TryParse(Marker3Offtime[index].Value, out offTime) == false)
{
throw new Exception(string.Format("Marker 3 Off Time Value of {0} couldn't be converted to an
integer.", Marker3Offtime[index].Value));
}

if (double.TryParse(Marker3Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 3 Attenuation Time Value of {0} couldn't be converted
to an integer.", Marker3Atten[index].Value));
}

mdata.Marker3Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation =
attenuation });
}

#endregion Marker 3
```

```
#region Marker 4

// Now Find the Marker 1 section.
//var marker4Section = from c in query.First().Descendants("section")
//                     where (string)c.Attribute("name") == "3"
//                     select c;

var Marker4Cont = from c in myList[4].Descendants("section")
where (string)c.Attribute("name") == "Transition"
select c;

// Now get ON Time.
var Marker4Ontime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OnTime"
select c).ToList();

// Now get Off Time
var Marker4Offtime = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "OffTime"
select c).ToList();

// Now Get Attenuation.
var Marker4Atten = (from c in Marker4Cont.Descendants("key")
where (string)c.Attribute("name") == "Attenuation"
select c).ToList();

// Convert Elements to  correct datatype.
for (int index = 0; index < Marker4Ontime.Count; index++)
{
int onTime;
int offTime;
double attenuation;

if (int.TryParse(Marker4Ontime[index].Value, out onTime) == false)
{
throw new Exception(string.Format("Marker 4 On Time Value of {0} couldn't be converted to an
integer.", Marker4Ontime[index].Value));
}

if (int.TryParse(Marker4Offtime[index].Value, out offTime) == false)
{
```

```
throw new Exception(string.Format("Marker 4 Off Time Value of {0} couldn't be converted to an integer.", Marker4Offtime[index].Value));
}

if (double.TryParse(Marker4Atten[index].Value, out attenuation) == false)
{
throw new Exception(string.Format("Marker 4 Attenuation Time Value of {0} couldn't be converted to an integer.", Marker4Atten[index].Value));
}


mdata.Marker4Info.Add(new Transition() { OnTime = onTime, OffTime = offTime, Attenuation = attenuation });
}

#endregion Marker 4

info.Marker = mdata;
#endif

// -------------------- Get number of slots --------------------

int numOfSlots = 0;

var querySubCarrierSpacing = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "CarrierDefinition"
select c;

var listSubCarrierSpacing = querySubCarrierSpacing.ToList();

// Get slot enable state
var subCarrierSpacing = (from c in listSubCarrierSpacing[0].Descendants("key")
where (string)c.Attribute("name") == "Subcarrier Spacing (Hz)"
select c).ToList();

// Convert elements to correct datatype
if (subCarrierSpacing[0].Value.Contains("15k")) numOfSlots = 10;
else if (subCarrierSpacing[0].Value.Contains("30k")) numOfSlots = 20;
else if (subCarrierSpacing[0].Value.Contains("60k")) numOfSlots = 40;
else if (subCarrierSpacing[0].Value.Contains("120k")) numOfSlots = 80;
else if (subCarrierSpacing[0].Value.Contains("240k")) numOfSlots = 160;
else if (subCarrierSpacing[0].Value.Contains("480k")) numOfSlots = 320;
```

```
else
{
throw new Exception(string.Format("Subcarrier spacing {0} couldn't be converted to an integer.",
subCarrierSpacing[0].Value));
}

// -------------------- Get slots enabled --------------------

var querySlotAllocation = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "PUSCH Slot Settings 0"
select c;

var listSlotAllocation = querySlotAllocation.ToList();

// Get list of enabled slots
var slotAllocation = (from c in listSlotAllocation[0].Descendants("key")
where (string)c.Attribute("name") == "Slot Allocation"
select c).ToList();

// Convert elements to correct datatype
string slotList = slotAllocation[0].Value;

info.SlotEnabled = new bool[numOfSlots];
for (int slotIndex = 0; slotIndex < numOfSlots; slotIndex++)
{
info.SlotEnabled[slotIndex] = false;
}

int numOfTriggers = 0;
int slotIndexStop = 0;
int slotIndexStart = 0;
char[] delimiter1 = { ',' };
char[] delimiter2 = { ':' };
double slotLength = 10e-3 / numOfSlots;

string[] slotsTemp1 = slotList.Split(delimiter1);

for (int i = 0; i < slotsTemp1.Length; i++)
{
string[] slotsTemp2 = slotsTemp1[i].Split(delimiter2);

slotIndexStart = Convert.ToInt32(slotsTemp2[0]);
```

```
if (slotsTemp2.Length > 1)
{
if (slotsTemp2[1].ToUpper() == "LAST")
{
slotIndexStop = numOfSlots - 1;
}
}
else
{
slotIndexStop = Convert.ToInt32(slotsTemp2[0]);
}

for (int slotIndex = slotIndexStart; slotIndex <= slotIndexStop; slotIndex++)
{
info.SlotEnabled[slotIndex] = true;
numOfTriggers++;
}
}

// -------------------- Get waveform markers --------------------

int onTime;
int offTime;
MarkerData mdata = new MarkerData();

for (int slotIndex = 0; slotIndex < numOfSlots; slotIndex++)
{
if (info.SlotEnabled[slotIndex] == true)
{
onTime = offTime = slotIndex;
mdata.Marker1Info.Add(new Transition() { OnTime = slotIndex, OffTime = slotIndex, Attenuation =
0 });
}
}

info.Marker = mdata;

// -------------------- Get bandwidth --------------------

var queryBandwidth = from c in xml.Root.Descendants("section")
where (string)c.Attribute("name") == "Cell Settings"
```

```
select c;

var listBandwidth = queryBandwidth.ToList();

// Get bandwidth
var bandwidth = (from c in listBandwidth[0].Descendants("key")
where (string)c.Attribute("name") == "Bandwidth (Hz)"
select c).ToList();

// Convert elements to correct datatype
if (bandwidth[0].Value.Contains("15M")) info.Bandwidth = ChannelBandwidth.FifteenMHz;
else if (bandwidth[0].Value.Contains("20M")) info.Bandwidth = ChannelBandwidth.TwentyMHz;
else if (bandwidth[0].Value.Contains("40M")) info.Bandwidth = ChannelBandwidth.FortyMHz;
else if (bandwidth[0].Value.Contains("50M")) info.Bandwidth = ChannelBandwidth.FiftyMHz;
else if (bandwidth[0].Value.Contains("60M")) info.Bandwidth = ChannelBandwidth.SixtyMHz;
else if (bandwidth[0].Value.Contains("80M")) info.Bandwidth = ChannelBandwidth.EightyMHz;
else if (bandwidth[0].Value.Contains("100M")) info.Bandwidth =
ChannelBandwidth.OneHundredMHz;
else if (bandwidth[0].Value.Contains("200M")) info.Bandwidth =
ChannelBandwidth.TwoHundredMHz;
else
{
throw new Exception(string.Format("NR bandwidth {0} is not valid.", bandwidth[0].Value));
}

return info;
}
}


//================================================================
==========
// The following classes / methods are reserved for backward compatibility.  Newer
// applications should use the code resources above.
//================================================================
==========

public class CalData
{
public CalData()
{
measCalFactor = new double[12];
```

```
measPath = new List<Measures>();
}

public string srcSelect = "";
public string srcPath = "";
public double srcFreq = 0;
public double srcLevel = -100;
public string modulationType = "";
public string modulationFile = "";
public double dutyCycle = 100;
public double calCalFactor = 0;
public double srcCalFactor = 0;
public double[] measCalFactor;
public List<Measures> measPath { get; set; }
}

public class CalDataNew
{
public int record = 0;
public string comment = "";
public string srcSelect = "";
public string srcPathMatrix = "";
public string srcPathRf = "";
public string measPathMatrix = "";
public string measPathRf = "";
public double srcFreq = 0;
public double srcLevel = -100;
public string modulationType = "";
public string modulationFile = "";
public double dutyCycle = 100;
public double calCalFactor = 0;
public double srcCalFactor = 0;
public double measCalFactor = 0;
public double measCalFactorBypass = 0;
public double noiseCalFactorSrc = 0;
public double noiseCalFactorSrcPath = 0;
public double noiseCalFactorOn = 0;
public double noiseCalFactorOff = 0;
public double noiseCalFactorLoss = 0;
public double noiseCalFactorENR = 0;
}
```

```
public class CalDataVer3
{
public int record = 0;
public bool copy = false;
public SigGen srcSelect;
public SrcPort srcPort;
public MeasPort measPort;
public MeasFilt measFilter;
public string comment = "";
public string srcPortAlias = "";
public string measPortAlias = "";
public string modulationType = "";
public string modulationFile = "";
public double srcFreq = 0;
public double srcLevel = -100;
public double dutyCycle = 100;
public double measAtten = 0;
public double[] srcCalFactor = new double[8];
public double[] measCalFactor = new double[8];
public double[] calCalFactor = new double[32];
public double noiseCalFactorSrc = 0;
public double noiseCalFactorSrcPath = 0;
public double noiseCalFactorOn = 0;
public double noiseCalFactorOff = 0;
public double noiseCalFactorLoss = 0;
public double noiseCalFactorENR = 0;

public double[] portModuleNoiseCalFactorOn = new double[4];
public double[] portModuleNoiseCalFactorOff = new double[4];
public double[] portModuleNoiseCalFactorENR = new double[4];
}

public class PowerMeter
{
public bool Available { get; set; }
}

public class InternalCalibration
{
public bool Available { get; set; }
}
```

```
public class Amplifier
{
public double gain = 0; // For backward compatibility

public double srcGain = 0;
public double measGain = 0;
}

public class ExternalGainLoss
{
public double srcGain = 0;
public double srcLoss = 0;
public double measGain = 0;
public double measLoss = 0;
}

public class Attenuation
{
public double[] srcAtten = new double[4];
public double[] measAtten = new double[12];
}

public class AttenuationNew
{
public double[] srcAtten;
public double[] measAtten;

public double[] srcAtten2;
public double[] measAtten2;
}

public class InternalAttenuation
{
public double[] srcAttenMatrix = new double[4];
public double[] measAttenMatrix = new double[12];

public double[] srcAttenRf = new double[12];
public double[] measAttenRf = new double[12];
}

public class ExternalAttenuation
{
```

```
public double[] srcAttenMatrix = new double[4];
public double[] measAttenMatrix = new double[12];

public double[] srcAttenRf = new double[12];
public double[] measAttenRf = new double[12];

public double[] srcAttenPortModule = new double[17];  // UserCal v3.0
public double[] measAttenPortModule = new double[46]; // UserCal v3.0
}

public class Measures
{
public string MeasureName { get; set; }
public string MeasurePopulated { get; set; }
}

public class Noise
{
public double[,] ENR = new double[2, 20];
public double[] noiseResult;
public double[] noiseResult2;
public double[] noiseOff;
public double[] noiseOn;
public double[] noiseLoss;
public double[] noiseENR;

public int numOfCalRecords { get; set; }

public void NoiseRecords(int numOfCalRecords)
{
this.numOfCalRecords = numOfCalRecords;
noiseResult = new double[numOfCalRecords];
noiseResult2 = new double[numOfCalRecords];
noiseOff = new double[numOfCalRecords];
noiseOn = new double[numOfCalRecords];
noiseLoss = new double[numOfCalRecords];
noiseENR = new double[numOfCalRecords];
}
}

public class NoiseData
{
```

```
public string srcSelect = "";
public string srcPortMatrix = "";
public string srcPortRf = "";
public string measPortMatrix = "";
public string measPortRf = "";

public string srcPort = "";  // For backward compatibility
public string measPort = ""; // For backward compatibility

public double freq = 0;
public double ENR = 0;
public double Loss = 0;
public double On = 0;
public double Off = 0;
}

public class CalImport
{
public Tuple<List<CalData>, PowerMeter, Amplifier, Attenuation> ImportCalConfig(string
calConfigFile)
{
int counter = 0;
bool flag = true;
string line = " ";
string[] splitLine = null;
Amplifier amp = new Amplifier();
PowerMeter usePowerMeter = new PowerMeter();
Attenuation attenuation = new Attenuation();
List<CalData> caldatum = new List<CalData>();

try
{
using (StreamReader stream = new StreamReader(calConfigFile))
{
Regex CSVParser = new Regex(",(?=(?:[^\"]*\"[^\"]*\")*(?![^\"]*\"))");

do
{
splitLine = CSVParser.Split(line);

if (flag)
{
```

```
// Header information is on rows 1 through 4 of the Cal_Config.csv file
for (int row = 0; row < 4; row++)
{
line = stream.ReadLine(); // Read rows 1 - 4
}

// Power meter starts on row 5 of the Cal_Config.csv file
line = stream.ReadLine();  // Read row 5
splitLine = CSVParser.Split(line);
usePowerMeter.Available = (char.Parse(splitLine[1]) == 'Y' ? true : false) ||
(char.Parse(splitLine[1]) == 'y' ? true : false);

// Amplifier gain starts on row 6 of the Cal_Config.csv file
line = stream.ReadLine();  // Read row 6
splitLine = CSVParser.Split(line);
amp.gain = double.Parse(splitLine[1]);

// Source path attenuation data starts on row 7 of the Cal_Config.csv file
line = stream.ReadLine();  // Read row 7
splitLine = CSVParser.Split(line);
for (int srcIndex = 0; srcIndex < 4; srcIndex++)
{
attenuation.srcAtten[srcIndex] = double.Parse(splitLine[srcIndex + 1]);
}

// Measure path attenuation data starts on row 8 of the Cal_Config.csv file
line = stream.ReadLine(); // Read row 8
splitLine = CSVParser.Split(line);
for (int measIndex = 0; measIndex < 12; measIndex++)
{
attenuation.measAtten[measIndex] = double.Parse(splitLine[measIndex + 1]);
}

// Calibration configuration data starts on row 11 of the Cal_Config.csv file
for (int row = 0; row < 3; row++)
{
line = stream.ReadLine(); // Read rows 9 - 11
}
splitLine = CSVParser.Split(line);

flag = false;
}
```

```
if (splitLine.Length > 1)
{
CalData caldata = new CalData();
caldata.srcSelect = splitLine[0];
caldata.srcPath = splitLine[1];
caldata.srcFreq = double.Parse(splitLine[2]);
caldata.srcLevel = double.Parse(splitLine[3]);
caldata.modulationType = splitLine[4];
caldata.modulationFile = splitLine[5];
caldata.dutyCycle = double.Parse(splitLine[6]);

counter = 1;

for (int measIndex = 9; measIndex < 21; measIndex++)
{
Measures measures = new Measures();
measures.MeasureName = "Meas" + counter.ToString();
measures.MeasurePopulated = splitLine[measIndex];

caldata.measPath.Add(measures);
counter++;
}

caldatum.Add(caldata);
}
} while ((line = stream.ReadLine()) != null);
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
}

return Tuple.Create(caldatum, usePowerMeter, amp, attenuation);
}

public Tuple<List<CalData>, PowerMeter, Amplifier, Attenuation, ExternalGainLoss>
ImportCalConfigNew(string calConfigFile)
{
string line = "";
PowerMeter usePowerMeter = new PowerMeter();
```

```
InternalCalibration useMatrixPath = new InternalCalibration();
Amplifier amp = new Amplifier(); // For backwards compatibility
ExternalGainLoss ext = new ExternalGainLoss();

Attenuation attenuation = new Attenuation();
AttenuationNew attenNew = new AttenuationNew();
List<CalData> caldatum = new List<CalData>();

try
{
using (StreamReader stream = new StreamReader(calConfigFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "POWER METER:") // For backwards compatibility
{
usePowerMeter.Available = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "USE EXTERNAL POWER METER:")
{
usePowerMeter.Available = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE INTERNAL MATRIX PATH:")
{
useMatrixPath.Available = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "AMPLIFIER GAIN:") // For backwards compatibility
{
amp.gain = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "SOURCE AMPLIFIER GAIN:")
{
ext.srcGain = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "SOURCE PATH LOSS:")
{
ext.srcLoss = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "MEASURE AMPLIFIER GAIN:")
{
```

```
ext.measGain = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "MEASURE PATH LOSS:")
{
ext.measLoss = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "SOURCE PATH ATTENUATORS:")
{
int srcIndex = 1;
int pathCount = 0;

while (line.Split(',')[srcIndex++] != "")
{
pathCount++; // Count number of source paths
}

attenNew.srcAtten = new double[pathCount];

for (srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenNew.srcAtten[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH ATTENUATORS:")
{
int measIndex = 1;
int pathCount = 0;

while (line.Split(',')[measIndex++] != "")
{
pathCount++; // Count number of measure paths
}

attenNew.measAtten = new double[pathCount];

for (measIndex = 0; measIndex < pathCount; measIndex++)
{
attenNew.measAtten[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG2" || tmpString.ToUpper() ==
"NOISE")
```

```
{
CalData caldata = new CalData();
caldata.srcSelect = line.Split(',')[0];
caldata.srcPath = line.Split(',')[1];
caldata.srcFreq = double.Parse(line.Split(',')[2]);
caldata.srcLevel = double.Parse(line.Split(',')[3]);
caldata.modulationType = line.Split(',')[4];
caldata.modulationFile = line.Split(',')[5];
caldata.dutyCycle = double.Parse(line.Split(',')[6]);

for (int measIndex = 9; measIndex < 21; measIndex++)
{
Measures measures = new Measures();
measures.MeasureName = "Meas" + (measIndex - 8).ToString();
measures.MeasurePopulated = line.Split(',')[measIndex];

caldata.measPath.Add(measures);
}

caldatum.Add(caldata);
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
}

return Tuple.Create(caldatum, usePowerMeter, amp, attenuation, ext);
}

public Tuple<List<CalData>, Attenuation> ImportCalData(string calDataFile)
{
int counter = 0;
bool flag = true;
string line = " ";
string[] splitLine = null;
Attenuation attenuation = new Attenuation();
List<CalData> caldatum = new List<CalData>();

try
```

```
{
using (StreamReader stream = new StreamReader(calDataFile))
{
Regex CSVParser = new Regex(",(?=(?:[^\"]*\"[^\"]*\")*(?![^\"]*\"))");

do
{
splitLine = CSVParser.Split(line);

if (flag)
{
// Header information is on rows 1 through 6 of the Cal_Config.csv file
for (int row = 0; row < 6; row++)
{
line = stream.ReadLine(); // Read rows 1 - 6
}

// Source path attenuation data starts on row 7 of the Cal_Config.csv file
line = stream.ReadLine();  // Read row 7
splitLine = CSVParser.Split(line);
for (int srcIndex = 0; srcIndex < 4; srcIndex++)
{
attenuation.srcAtten[srcIndex] = double.Parse(splitLine[srcIndex + 1]);
}

// Measure path attenuation data starts on row 8 of the Cal_Config.csv file
line = stream.ReadLine(); // Read row 8
splitLine = CSVParser.Split(line);
for (int measIndex = 0; measIndex < 12; measIndex++)
{
attenuation.measAtten[measIndex] = double.Parse(splitLine[measIndex + 1]);
}

// Calibration configuration data starts on row 11 of the Cal_Config.csv file
for (int row = 0; row < 3; row++)
{
line = stream.ReadLine(); // Read rows 9 - 11
}
splitLine = CSVParser.Split(line);

flag = false;
}
```

```
if (splitLine.Length > 1)
{
CalData caldata = new CalData();
caldata.srcSelect = splitLine[0];
caldata.srcPath = splitLine[1];
caldata.srcFreq = double.Parse(splitLine[2]);
caldata.srcLevel = double.Parse(splitLine[3]);
caldata.modulationType = splitLine[4];
caldata.modulationFile = splitLine[5];
caldata.dutyCycle = double.Parse(splitLine[6]);
caldata.calCalFactor = double.Parse(splitLine[7]);
caldata.srcCalFactor = double.Parse(splitLine[8]);

counter = 1;

for (int measIndex = 9; measIndex < 21; measIndex++)
{
//Measures measures = new Measures();
//measures.MeasureName = "Meas" + counter.ToString();
//measures.MeasurePopulated = splitLine[measIndex];

//caldata.measPath.Add(measures);

caldata.measCalFactor[measIndex - 9] = double.Parse(splitLine[measIndex]);
counter++;
}

caldatum.Add(caldata);
}

} while ((line = stream.ReadLine()) != null);
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
}

return Tuple.Create(caldatum, attenuation);
}
```

```
public Tuple<List<CalDataNew>, AttenuationNew> ImportCalDataNew2(string calDataFile)
{
int counter = 0;
bool flag = true;
string line = " ";
string[] splitLine = null;
List<CalDataNew> caldatum = new List<CalDataNew>();
AttenuationNew attenuation = new AttenuationNew();
attenuation.srcAtten = new double[4];
attenuation.measAtten = new double[12];
attenuation.srcAtten2 = new double[4];
attenuation.measAtten2 = new double[12];

try
{
using (StreamReader stream = new StreamReader(calDataFile))
{
Regex CSVParser = new Regex(",(?=(?:[^\"]*\"[^\"]*\")*(?![^\"]*\"))");

do
{
splitLine = CSVParser.Split(line);

if (flag)
{
// Header information is on rows 1 through 7 of the Cal_Config.csv file
for (int row = 0; row < 7; row++)
{
line = stream.ReadLine(); // Read rows 1 - 7
}

// Matrix source path attenuation data starts on row 8 of the Cal_Config.csv file
line = stream.ReadLine(); // Read row 8
splitLine = CSVParser.Split(line);
for (int srcIndex = 0; srcIndex < 4; srcIndex++)
{
attenuation.srcAtten[srcIndex] = double.Parse(splitLine[srcIndex + 1]);
}

// Matrix measure path attenuation data starts on row 9 of the Cal_Config.csv file
line = stream.ReadLine(); // Read row 9
splitLine = CSVParser.Split(line);
```

```
for (int measIndex = 0; measIndex < 12; measIndex++)
{
attenuation.measAtten[measIndex] = double.Parse(splitLine[measIndex + 1]);
}

line = stream.ReadLine(); // Read row 10 (blank)

// RF source path attenuation data starts on row 11 of the Cal_Config.csv file
line = stream.ReadLine(); // Read row 11
splitLine = CSVParser.Split(line);
for (int srcIndex = 0; srcIndex < 4; srcIndex++)
{
attenuation.srcAtten2[srcIndex] = double.Parse(splitLine[srcIndex + 1]);
}

// RF measure path attenuation data starts on row 12 of the Cal_Config.csv file
line = stream.ReadLine(); // Read row 12
splitLine = CSVParser.Split(line);
for (int measIndex = 0; measIndex < 12; measIndex++)
{
attenuation.measAtten2[measIndex] = double.Parse(splitLine[measIndex + 1]);
}

line = stream.ReadLine(); // Read row 13 (blank)
line = stream.ReadLine(); // Read row 14 (header)
line = stream.ReadLine(); // Read row 15 (data start)
splitLine = CSVParser.Split(line);

flag = false;
}

if (splitLine.Length > 1)
{
CalDataNew caldata = new CalDataNew();
caldata.srcSelect = splitLine[0];
caldata.srcPathMatrix = splitLine[1];
caldata.srcPathRf = splitLine[2];
caldata.measPathMatrix = splitLine[3];
caldata.measPathRf = splitLine[4];
caldata.srcFreq = double.Parse(splitLine[5]);
caldata.srcLevel = double.Parse(splitLine[6]);
caldata.modulationType = splitLine[7];
```

```
caldata.modulationFile = splitLine[8];
caldata.dutyCycle = double.Parse(splitLine[9]);
caldata.comment = splitLine[10];
caldata.calCalFactor = double.Parse(splitLine[11]);
caldata.srcCalFactor = double.Parse(splitLine[12]);
caldata.measCalFactor = double.Parse(splitLine[13]);

counter++;

caldatum.Add(caldata);
}

} while ((line = stream.ReadLine()) != null);
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
}

return Tuple.Create(caldatum, attenuation);
}

public Tuple<List<CalDataNew>, Info, CalSettings, Amplifier, InternalAttenuation,
ExternalAttenuation> ImportCalConfigVer2(string calConfigFile)
{
string line = "";
Info info = new Info();
CalSettings CalSelect = new CalSettings();
Amplifier amp = new Amplifier();
InternalAttenuation attenInternal = new InternalAttenuation();
ExternalAttenuation attenExternal = new ExternalAttenuation();
List<CalDataNew> caldatum = new List<CalDataNew>();

try
{
using (StreamReader stream = new StreamReader(calConfigFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];
```

```
if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
if (tmpString.ToUpper() == "REVISION:")
{
info.Revision = line.Split(',')[1];
}
if (tmpString.ToUpper() == "TEST PROGRAM:")
{
info.Program = line.Split(',')[1];
}
if (tmpString.ToUpper() == "COMMENT:")
{
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "USE EXTERNAL POWER METER:")
{
CalSelect.PowerMeterAvailable = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE RF SOURCE/MEASURE:")
{
CalSelect.SourceMeasurePath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE MEASURE PATH LNA:")
{
CalSelect.MeasurePathLNA = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE INTERNAL MATRIX PATH:")
{
CalSelect.InternalMatrixPath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE NOISE SOURCE:")
{
CalSelect.NoiseSource = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE DC SENSE:")
{
CalSelect.DCSense = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "SOURCE AMPLIFIER GAIN:")
{
```

```
amp.srcGain = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "MEASURE AMPLIFIER GAIN:")
{
amp.measGain = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "SOURCE PATH LOSS (MATRIX):")
{
int pathCount = 4;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenInternal.srcAttenMatrix[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH LOSS (MATRIX):")
{
int pathCount = 12;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenInternal.measAttenMatrix[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SOURCE PATH LOSS (RF):")
{
int pathCount = 12;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenInternal.srcAttenRf[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH LOSS (RF):")
{
int pathCount = 12;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenInternal.measAttenRf[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
```

```
else if (tmpString.ToUpper() == "SOURCE PATH ATTENUATORS (MATRIX):")
{
int pathCount = 4;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenExternal.srcAttenMatrix[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH ATTENUATORS (MATRIX):")
{
int pathCount = 12;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenExternal.measAttenMatrix[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SOURCE PATH ATTENUATORS (RF):")
{
int pathCount = 12;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenExternal.srcAttenRf[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH ATTENUATORS (RF):")
{
int pathCount = 12;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenExternal.measAttenRf[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG2" || tmpString.ToUpper() ==
"NOISE")
{
CalDataNew caldata = new CalDataNew();
caldata.srcSelect = line.Split(',')[0];
caldata.srcPathMatrix = line.Split(',')[1];
```

```
caldata.srcPathRf = line.Split(',')[2];
caldata.measPathMatrix = line.Split(',')[3];
caldata.measPathRf = line.Split(',')[4];
caldata.srcFreq = double.Parse(line.Split(',')[5]);
caldata.srcLevel = double.Parse(line.Split(',')[6]);
caldata.modulationType = line.Split(',')[7];
caldata.modulationFile = line.Split(',')[8];
caldata.dutyCycle = double.Parse(line.Split(',')[9]);
caldata.comment = line.Split(',')[10];

caldatum.Add(caldata);
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
}

return Tuple.Create(caldatum, info, CalSelect, amp, attenInternal, attenExternal);
}

public Tuple<List<CalDataNew>, Info, PowerMeter, Amplifier, InternalAttenuation,
ExternalAttenuation, InternalCalibration> ImportCalDataVer2(string calDataFile)
{
string line = "";
Info info = new Info();
PowerMeter usePowerMeter = new PowerMeter();
InternalCalibration useMatrixPath = new InternalCalibration();
Amplifier amp = new Amplifier();
InternalAttenuation attenInternal = new InternalAttenuation();
ExternalAttenuation attenExternal = new ExternalAttenuation();
List<CalDataNew> caldatum = new List<CalDataNew>();

try
{
using (StreamReader stream = new StreamReader(calDataFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];
```

```
if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
if (tmpString.ToUpper() == "REVISION:")
{
info.Revision = line.Split(',')[1];
}
if (tmpString.ToUpper() == "TEST PROGRAM:")
{
info.Program = line.Split(',')[1];
}
if (tmpString.ToUpper() == "COMMENT:")
{
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "USE EXTERNAL POWER METER:")
{
usePowerMeter.Available = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "SOURCE AMPLIFIER GAIN:")
{
amp.srcGain = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "MEASURE AMPLIFIER GAIN:")
{
amp.measGain = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "SOURCE PATH LOSS (MATRIX):")
{
int pathCount = 4;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenInternal.srcAttenMatrix[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH LOSS (MATRIX):")
{
int pathCount = 12;
```

```
for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenInternal.measAttenMatrix[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SOURCE PATH LOSS (RF):")
{
int pathCount = 12;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenInternal.srcAttenRf[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH LOSS (RF):")
{
int pathCount = 12;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenInternal.measAttenRf[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SOURCE PATH ATTENUATORS (MATRIX):")
{
int pathCount = 4;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenExternal.srcAttenMatrix[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH ATTENUATORS (MATRIX):")
{
int pathCount = 12;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenExternal.measAttenMatrix[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SOURCE PATH ATTENUATORS (RF):")
```

```
{
int pathCount = 12;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenExternal.srcAttenRf[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH ATTENUATORS (RF):")
{
int pathCount = 12;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenExternal.measAttenRf[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG2" || tmpString.ToUpper() ==
"NOISE")
{
CalDataNew caldata = new CalDataNew();
caldata.srcSelect = line.Split(',')[0];
caldata.srcPathMatrix = line.Split(',')[1];
caldata.srcPathRf = line.Split(',')[2];
caldata.measPathMatrix = line.Split(',')[3];
caldata.measPathRf = line.Split(',')[4];
caldata.srcFreq = double.Parse(line.Split(',')[5]);
caldata.srcLevel = double.Parse(line.Split(',')[6]);
caldata.modulationType = line.Split(',')[7];
caldata.modulationFile = line.Split(',')[8];
caldata.dutyCycle = double.Parse(line.Split(',')[9]);
caldata.comment = line.Split(',')[10];
caldata.calCalFactor = double.Parse(line.Split(',')[11]);
caldata.srcCalFactor = double.Parse(line.Split(',')[12]);
caldata.measCalFactor = double.Parse(line.Split(',')[13]);
caldata.measCalFactorBypass = double.Parse(line.Split(',')[14]);

caldatum.Add(caldata);
}
}
}
}
```

```
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
}

return Tuple.Create(caldatum, info, usePowerMeter, amp, attenInternal, attenExternal,
useMatrixPath);
}

public Tuple<List<CalDataVer3>, Info, CalSettings, ExternalAttenuation>
ImportCalConfigVer3(string calConfigFile)
{
string line = "";
Info info = new Info();
CalSettings CalSelect = new CalSettings();
ExternalAttenuation attenExternal = new ExternalAttenuation();
List<CalDataVer3> caldatum = new List<CalDataVer3>();

try
{
using (StreamReader stream = new StreamReader(calConfigFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
if (tmpString.ToUpper() == "REVISION:")
{
info.Revision = line.Split(',')[1];
}
if (tmpString.ToUpper() == "TEST PROGRAM:")
{
info.Program = line.Split(',')[1];
}
if (tmpString.ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = line.Split(',')[1];
}
```

```
if (tmpString.ToUpper() == "COMMENT:")
{
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "USE EXTERNAL POWER METER:")
{
CalSelect.PowerMeterAvailable = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE RF SOURCE/MEASURE:")
{
CalSelect.SourceMeasurePath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE INTERNAL DIRECT PATH:")
{
CalSelect.InternalMatrixPath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE NOISE SOURCE:")
{
CalSelect.NoiseSource = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE DC SENSE:")
{
CalSelect.DCSense = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "SOURCE PATH ATTENUATORS:")
{
const int pathCount = 17;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenExternal.srcAttenPortModule[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH ATTENUATORS:")
{
const int pathCount = 46;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenExternal.measAttenPortModule[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
```

```
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG2" || tmpString.ToUpper() ==
"NOISE")
{
CalDataVer3 caldata = new CalDataVer3();

tmpString = line.Split(',')[0];
if (tmpString.ToUpper() == "SG1") caldata.srcSelect = SigGen.SG1;
else if (tmpString.ToUpper() == "SG2") caldata.srcSelect = SigGen.SG2;
else if (tmpString.ToUpper() == "NOISE") caldata.srcSelect = SigGen.NOISE;

tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") caldata.srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") caldata.srcPort = SrcPort.P2;
else if (tmpString.ToUpper() == "P3") caldata.srcPort = SrcPort.P3;
else if (tmpString.ToUpper() == "P4") caldata.srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") caldata.srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") caldata.srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") caldata.srcPort = SrcPort.P7;
else if (tmpString.ToUpper() == "P8") caldata.srcPort = SrcPort.P8;
else if (tmpString.ToUpper() == "P9") caldata.srcPort = SrcPort.P9;
else if (tmpString.ToUpper() == "P10") caldata.srcPort = SrcPort.P10;
else if (tmpString.ToUpper() == "P11") caldata.srcPort = SrcPort.P11;
else if (tmpString.ToUpper() == "P12") caldata.srcPort = SrcPort.P12;
else if (tmpString.ToUpper() == "P13") caldata.srcPort = SrcPort.P13;
else if (tmpString.ToUpper() == "P14") caldata.srcPort = SrcPort.P14;
else if (tmpString.ToUpper() == "P15") caldata.srcPort = SrcPort.P15;
else if (tmpString.ToUpper() == "P16") caldata.srcPort = SrcPort.P16;
else if (tmpString.ToUpper() == "SG1") caldata.srcPort = SrcPort.SG1_OUT;
else if (tmpString.ToUpper() == "CAL") caldata.srcPort = SrcPort.CAL;
else caldata.srcPort = SrcPort.NONE;

caldata.srcPortAlias = line.Split(',')[2];

tmpString = line.Split(',')[3];
if (tmpString.ToUpper() == "P1") caldata.measPort = MeasPort.P1;
else if (tmpString.ToUpper() == "P2") caldata.measPort = MeasPort.P2;
else if (tmpString.ToUpper() == "P3") caldata.measPort = MeasPort.P3;
else if (tmpString.ToUpper() == "P4") caldata.measPort = MeasPort.P4;
else if (tmpString.ToUpper() == "P5") caldata.measPort = MeasPort.P5;
else if (tmpString.ToUpper() == "P6") caldata.measPort = MeasPort.P6;
else if (tmpString.ToUpper() == "P7") caldata.measPort = MeasPort.P7;
else if (tmpString.ToUpper() == "P8") caldata.measPort = MeasPort.P8;
```

```
else if (tmpString.ToUpper() == "P9") caldata.measPort = MeasPort.P9;
else if (tmpString.ToUpper() == "P10") caldata.measPort = MeasPort.P10;
else if (tmpString.ToUpper() == "P11") caldata.measPort = MeasPort.P11;
else if (tmpString.ToUpper() == "P12") caldata.measPort = MeasPort.P12;
else if (tmpString.ToUpper() == "P13") caldata.measPort = MeasPort.P13;
else if (tmpString.ToUpper() == "P14") caldata.measPort = MeasPort.P14;
else if (tmpString.ToUpper() == "P15") caldata.measPort = MeasPort.P15;
else if (tmpString.ToUpper() == "P16") caldata.measPort = MeasPort.P16;
else if (tmpString.ToUpper() == "P1_FWD") caldata.measPort = MeasPort.P1_Fwd;
else if (tmpString.ToUpper() == "P2_FWD") caldata.measPort = MeasPort.P2_Fwd;
else if (tmpString.ToUpper() == "P3_FWD") caldata.measPort = MeasPort.P3_Fwd;
else if (tmpString.ToUpper() == "P4_FWD") caldata.measPort = MeasPort.P4_Fwd;
else if (tmpString.ToUpper() == "P5_FWD") caldata.measPort = MeasPort.P5_Fwd;
else if (tmpString.ToUpper() == "P6_FWD") caldata.measPort = MeasPort.P6_Fwd;
else if (tmpString.ToUpper() == "P7_FWD") caldata.measPort = MeasPort.P7_Fwd;
else if (tmpString.ToUpper() == "P8_FWD") caldata.measPort = MeasPort.P8_Fwd;
else if (tmpString.ToUpper() == "P1_REV") caldata.measPort = MeasPort.P1_Rev;
else if (tmpString.ToUpper() == "P2_REV") caldata.measPort = MeasPort.P2_Rev;
else if (tmpString.ToUpper() == "P3_REV") caldata.measPort = MeasPort.P3_Rev;
else if (tmpString.ToUpper() == "P4_REV") caldata.measPort = MeasPort.P4_Rev;
else if (tmpString.ToUpper() == "P5_REV") caldata.measPort = MeasPort.P5_Rev;
else if (tmpString.ToUpper() == "P6_REV") caldata.measPort = MeasPort.P6_Rev;
else if (tmpString.ToUpper() == "P7_REV") caldata.measPort = MeasPort.P7_Rev;
else if (tmpString.ToUpper() == "P8_REV") caldata.measPort = MeasPort.P8_Rev;
else if (tmpString.ToUpper() == "M1") caldata.measPort = MeasPort.M1;
else if (tmpString.ToUpper() == "M2") caldata.measPort = MeasPort.M2;
else if (tmpString.ToUpper() == "M3") caldata.measPort = MeasPort.M3;
else if (tmpString.ToUpper() == "M4") caldata.measPort = MeasPort.M4;
else if (tmpString.ToUpper() == "M5") caldata.measPort = MeasPort.M5;
else if (tmpString.ToUpper() == "M6") caldata.measPort = MeasPort.M6;
else if (tmpString.ToUpper() == "M7") caldata.measPort = MeasPort.M7;
else if (tmpString.ToUpper() == "M8") caldata.measPort = MeasPort.M8;
else if (tmpString.ToUpper() == "M9") caldata.measPort = MeasPort.M9;
else if (tmpString.ToUpper() == "M10") caldata.measPort = MeasPort.M10;
else if (tmpString.ToUpper() == "M11") caldata.measPort = MeasPort.M11;
else if (tmpString.ToUpper() == "M12") caldata.measPort = MeasPort.M12;
else if (tmpString.ToUpper() == "M13") caldata.measPort = MeasPort.M13;
else if (tmpString.ToUpper() == "M14") caldata.measPort = MeasPort.M14;
else if (tmpString.ToUpper() == "M15") caldata.measPort = MeasPort.M15;
else if (tmpString.ToUpper() == "M16") caldata.measPort = MeasPort.M16;
else if (tmpString.ToUpper() == "M17") caldata.measPort = MeasPort.M17;
else if (tmpString.ToUpper() == "M18") caldata.measPort = MeasPort.M18;
```

```
else if (tmpString.ToUpper() == "M19") caldata.measPort = MeasPort.M19;
else if (tmpString.ToUpper() == "M20") caldata.measPort = MeasPort.M20;
else if (tmpString.ToUpper() == "M21") caldata.measPort = MeasPort.M21;
else if (tmpString.ToUpper() == "M22") caldata.measPort = MeasPort.M22;
else if (tmpString.ToUpper() == "M23") caldata.measPort = MeasPort.M23;
else if (tmpString.ToUpper() == "M24") caldata.measPort = MeasPort.M24;
else if (tmpString.ToUpper() == "M25") caldata.measPort = MeasPort.M25;
else if (tmpString.ToUpper() == "M26") caldata.measPort = MeasPort.M26;
else if (tmpString.ToUpper() == "MA") caldata.measPort = MeasPort.MA;
else if (tmpString.ToUpper() == "MB") caldata.measPort = MeasPort.MB;
else if (tmpString.ToUpper() == "MC") caldata.measPort = MeasPort.MC;
else if (tmpString.ToUpper() == "MD") caldata.measPort = MeasPort.MD;
else if (tmpString.ToUpper() == "NONE") caldata.measPort = MeasPort.NONE;
else caldata.measPort = MeasPort.NONE;

caldata.measPortAlias = line.Split(',')[4];
caldata.srcFreq = double.Parse(line.Split(',')[5]);
caldata.srcLevel = double.Parse(line.Split(',')[6]);

tmpString = line.Split(',')[7];
if (tmpString.ToUpper() == "BYPASS") caldata.measFilter = MeasFilt.Bypass;
else if (tmpString.ToUpper() == "ATTEN") caldata.measFilter = MeasFilt.Atten;
else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1") caldata.measFilter =
MeasFilt.FILT1;
else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2") caldata.measFilter =
MeasFilt.FILT2;
else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3") caldata.measFilter =
MeasFilt.FILT3;
else caldata.measFilter = MeasFilt.Bypass; // Default

caldata.measAtten = double.Parse(line.Split(',')[8]);
caldata.modulationType = line.Split(',')[9];
caldata.modulationFile = line.Split(',')[10];
caldata.dutyCycle = double.Parse(line.Split(',')[11]);
caldata.comment = line.Split(',')[12];

caldatum.Add(caldata);
}
}
}
}
catch (FileNotFoundException error)
```

```
{
Console.WriteLine("\n{0}", error.Message);
}

return Tuple.Create(caldatum, info, CalSelect, attenExternal);
}

public Tuple<List<CalDataVer3>, Info, CalSettings, ExternalAttenuation>
ImportCalDataVer3(string calDataFile)
{
string line = "";
Info info = new Info();
CalSettings CalSelect = new CalSettings();
ExternalAttenuation attenExternal = new ExternalAttenuation();
List<CalDataVer3> caldatum = new List<CalDataVer3>();

try
{
using (StreamReader stream = new StreamReader(calDataFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
if (tmpString.ToUpper() == "REVISION:")
{
info.Revision = line.Split(',')[1];
}
if (tmpString.ToUpper() == "TEST PROGRAM:")
{
info.Program = line.Split(',')[1];
}
if (tmpString.ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = line.Split(',')[1];
}
if (tmpString.ToUpper() == "COMMENT:")
{
```

```
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "USE EXTERNAL POWER METER:")
{
CalSelect.PowerMeterAvailable = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE RF SOURCE/MEASURE:")
{
CalSelect.SourceMeasurePath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE INTERNAL MATRIX PATH:")
{
CalSelect.InternalMatrixPath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE NOISE SOURCE:")
{
CalSelect.NoiseSource = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE DC SENSE:")
{
CalSelect.DCSense = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "SOURCE PATH ATTENUATORS:")
{
const int pathCount = 17;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenExternal.srcAttenPortModule[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MEASURE PATH ATTENUATORS:")
{
const int pathCount = 46;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenExternal.measAttenPortModule[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG2" || tmpString.ToUpper() ==
"NOISE")
```

```
{
CalDataVer3 caldata = new CalDataVer3();

tmpString = line.Split(',')[0];
if (tmpString.ToUpper() == "SG1") caldata.srcSelect = SigGen.SG1;
else if (tmpString.ToUpper() == "SG2") caldata.srcSelect = SigGen.SG2;

tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") caldata.srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") caldata.srcPort = SrcPort.P2;
else if (tmpString.ToUpper() == "P3") caldata.srcPort = SrcPort.P3;
else if (tmpString.ToUpper() == "P4") caldata.srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") caldata.srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") caldata.srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") caldata.srcPort = SrcPort.P7;
else if (tmpString.ToUpper() == "P8") caldata.srcPort = SrcPort.P8;
else if (tmpString.ToUpper() == "P9") caldata.srcPort = SrcPort.P9;
else if (tmpString.ToUpper() == "P10") caldata.srcPort = SrcPort.P10;
else if (tmpString.ToUpper() == "P11") caldata.srcPort = SrcPort.P11;
else if (tmpString.ToUpper() == "P12") caldata.srcPort = SrcPort.P12;
else if (tmpString.ToUpper() == "P13") caldata.srcPort = SrcPort.P13;
else if (tmpString.ToUpper() == "P14") caldata.srcPort = SrcPort.P14;
else if (tmpString.ToUpper() == "P15") caldata.srcPort = SrcPort.P15;
else if (tmpString.ToUpper() == "P16") caldata.srcPort = SrcPort.P16;
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG1_OUT") caldata.srcPort =
SrcPort.SG1_OUT;
else if (tmpString.ToUpper() == "CAL") caldata.srcPort = SrcPort.CAL;
else caldata.srcPort = SrcPort.NONE;

caldata.srcPortAlias = line.Split(',')[2];

tmpString = line.Split(',')[3];
if (tmpString.ToUpper() == "P1") caldata.measPort = MeasPort.P1;
else if (tmpString.ToUpper() == "P2") caldata.measPort = MeasPort.P2;
else if (tmpString.ToUpper() == "P3") caldata.measPort = MeasPort.P3;
else if (tmpString.ToUpper() == "P4") caldata.measPort = MeasPort.P4;
else if (tmpString.ToUpper() == "P5") caldata.measPort = MeasPort.P5;
else if (tmpString.ToUpper() == "P6") caldata.measPort = MeasPort.P6;
else if (tmpString.ToUpper() == "P7") caldata.measPort = MeasPort.P7;
else if (tmpString.ToUpper() == "P8") caldata.measPort = MeasPort.P8;
else if (tmpString.ToUpper() == "P9") caldata.measPort = MeasPort.P9;
else if (tmpString.ToUpper() == "P10") caldata.measPort = MeasPort.P10;
```

```
else if (tmpString.ToUpper() == "P11") caldata.measPort = MeasPort.P11;
else if (tmpString.ToUpper() == "P12") caldata.measPort = MeasPort.P12;
else if (tmpString.ToUpper() == "P13") caldata.measPort = MeasPort.P13;
else if (tmpString.ToUpper() == "P14") caldata.measPort = MeasPort.P14;
else if (tmpString.ToUpper() == "P15") caldata.measPort = MeasPort.P15;
else if (tmpString.ToUpper() == "P16") caldata.measPort = MeasPort.P16;
else if (tmpString.ToUpper() == "P1_FWD") caldata.measPort = MeasPort.P1_Fwd;
else if (tmpString.ToUpper() == "P2_FWD") caldata.measPort = MeasPort.P2_Fwd;
else if (tmpString.ToUpper() == "P3_FWD") caldata.measPort = MeasPort.P3_Fwd;
else if (tmpString.ToUpper() == "P4_FWD") caldata.measPort = MeasPort.P4_Fwd;
else if (tmpString.ToUpper() == "P5_FWD") caldata.measPort = MeasPort.P5_Fwd;
else if (tmpString.ToUpper() == "P6_FWD") caldata.measPort = MeasPort.P6_Fwd;
else if (tmpString.ToUpper() == "P7_FWD") caldata.measPort = MeasPort.P7_Fwd;
else if (tmpString.ToUpper() == "P8_FWD") caldata.measPort = MeasPort.P8_Fwd;
else if (tmpString.ToUpper() == "P1_REV") caldata.measPort = MeasPort.P1_Rev;
else if (tmpString.ToUpper() == "P2_REV") caldata.measPort = MeasPort.P2_Rev;
else if (tmpString.ToUpper() == "P3_REV") caldata.measPort = MeasPort.P3_Rev;
else if (tmpString.ToUpper() == "P4_REV") caldata.measPort = MeasPort.P4_Rev;
else if (tmpString.ToUpper() == "P5_REV") caldata.measPort = MeasPort.P5_Rev;
else if (tmpString.ToUpper() == "P6_REV") caldata.measPort = MeasPort.P6_Rev;
else if (tmpString.ToUpper() == "P7_REV") caldata.measPort = MeasPort.P7_Rev;
else if (tmpString.ToUpper() == "P8_REV") caldata.measPort = MeasPort.P8_Rev;
else if (tmpString.ToUpper() == "M1") caldata.measPort = MeasPort.M1;
else if (tmpString.ToUpper() == "M2") caldata.measPort = MeasPort.M2;
else if (tmpString.ToUpper() == "M3") caldata.measPort = MeasPort.M3;
else if (tmpString.ToUpper() == "M4") caldata.measPort = MeasPort.M4;
else if (tmpString.ToUpper() == "M5") caldata.measPort = MeasPort.M5;
else if (tmpString.ToUpper() == "M6") caldata.measPort = MeasPort.M6;
else if (tmpString.ToUpper() == "M7") caldata.measPort = MeasPort.M7;
else if (tmpString.ToUpper() == "M8") caldata.measPort = MeasPort.M8;
else if (tmpString.ToUpper() == "M9") caldata.measPort = MeasPort.M9;
else if (tmpString.ToUpper() == "M10") caldata.measPort = MeasPort.M10;
else if (tmpString.ToUpper() == "M11") caldata.measPort = MeasPort.M11;
else if (tmpString.ToUpper() == "M12") caldata.measPort = MeasPort.M12;
else if (tmpString.ToUpper() == "M13") caldata.measPort = MeasPort.M13;
else if (tmpString.ToUpper() == "M14") caldata.measPort = MeasPort.M14;
else if (tmpString.ToUpper() == "M15") caldata.measPort = MeasPort.M15;
else if (tmpString.ToUpper() == "M16") caldata.measPort = MeasPort.M16;
else if (tmpString.ToUpper() == "M17") caldata.measPort = MeasPort.M17;
else if (tmpString.ToUpper() == "M18") caldata.measPort = MeasPort.M18;
else if (tmpString.ToUpper() == "M19") caldata.measPort = MeasPort.M19;
else if (tmpString.ToUpper() == "M20") caldata.measPort = MeasPort.M20;
```

```
else if (tmpString.ToUpper() == "M21") caldata.measPort = MeasPort.M21;
else if (tmpString.ToUpper() == "M22") caldata.measPort = MeasPort.M22;
else if (tmpString.ToUpper() == "M23") caldata.measPort = MeasPort.M23;
else if (tmpString.ToUpper() == "M24") caldata.measPort = MeasPort.M24;
else if (tmpString.ToUpper() == "M25") caldata.measPort = MeasPort.M25;
else if (tmpString.ToUpper() == "M26") caldata.measPort = MeasPort.M26;
else if (tmpString.ToUpper() == "MA") caldata.measPort = MeasPort.MA;
else if (tmpString.ToUpper() == "MB") caldata.measPort = MeasPort.MB;
else if (tmpString.ToUpper() == "MC") caldata.measPort = MeasPort.MC;
else if (tmpString.ToUpper() == "MD") caldata.measPort = MeasPort.MD;
else if (tmpString.ToUpper() == "NONE") caldata.measPort = MeasPort.NONE;
else caldata.measPort = MeasPort.NONE;

caldata.measPortAlias = line.Split(',')[4];
caldata.srcFreq = double.Parse(line.Split(',')[5]);
caldata.srcLevel = double.Parse(line.Split(',')[6]);

tmpString = line.Split(',')[7];
if (tmpString.ToUpper() == "BYPASS") caldata.measFilter = MeasFilt.Bypass;
else if (tmpString.ToUpper() == "ATTEN") caldata.measFilter = MeasFilt.Atten;
else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1") caldata.measFilter =
MeasFilt.FILT1;
else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2") caldata.measFilter =
MeasFilt.FILT2;
else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3") caldata.measFilter =
MeasFilt.FILT3;
else caldata.measFilter = MeasFilt.Bypass; // Default

caldata.measAtten = double.Parse(line.Split(',')[8]);
caldata.modulationType = line.Split(',')[9];
caldata.modulationFile = line.Split(',')[10];
caldata.dutyCycle = double.Parse(line.Split(',')[11]);
caldata.comment = line.Split(',')[12];

for (int i = 0; i < 8; i++)
{
caldata.srcCalFactor[i] = double.Parse(line.Split(',')[i + 13]);
}

for (int i = 0; i < 4; i++)
{
caldata.measCalFactor[i] = double.Parse(line.Split(',')[i + 21]);
```

```
}

for (int i = 0; i < 4; i++)
{

caldata.calCalFactor[i] = double.Parse(line.Split(',')[i + 25]);
}

caldatum.Add(caldata);
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
}

return Tuple.Create(caldatum, info, CalSelect, attenExternal);
}

public Tuple<Info, CalSettings, ExternalAttenuation, List<CalDataPortModule>>
ImportCalConfigVer4(string calConfigFile)
{
string line = "";
Info info = new Info();
CalSettings CalSelect = new CalSettings();
ExternalAttenuation attenExternal = new ExternalAttenuation();
List<CalDataPortModule> calDatum = new List<CalDataPortModule>();

// Initialize CalSelect
CalSelect.PowerMeterAvailable = true;
CalSelect.SourceMeasurePath = false;
CalSelect.InternalMatrixPath = true;
CalSelect.NoiseSource = false;
CalSelect.VNA = true;
CalSelect.NoiseBandwidth = 8e6;

try
{
using (StreamReader stream = new StreamReader(calConfigFile))
{
```

```
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
if (tmpString.ToUpper() == "REVISION:")
{
info.Revision = line.Split(',')[1];
}
if (tmpString.ToUpper() == "TEST PROGRAM:")
{
info.Program = line.Split(',')[1];
}
if (tmpString.ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = line.Split(',')[1];
}
if (tmpString.ToUpper() == "COMMENT:")
{
info.Comment = line.Split(',')[1];
}
//else if (tmpString.ToUpper() == "USE EXTERNAL POWER METER:")
//{
//    CalSelect.PowerMeterAvailable = line.Split(',')[1].ToUpper() == "Y" ? true : false;
//}
//else if (tmpString.ToUpper() == "CALIBRATE INTERNAL DIRECT PATH:")
//{
//    CalSelect.InternalMatrixPath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
//}
//else if (tmpString.ToUpper() == "CALIBRATE DC SENSE:")
//{
//    CalSelect.DCSense = line.Split(',')[1].ToUpper() == "Y" ? true : false;
//}
else if (tmpString.ToUpper() == "CALIBRATE RF SOURCE/MEASURE:")
{
CalSelect.SourceMeasurePath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE NOISE SOURCE:")
{
```

```
CalSelect.NoiseSource = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE VNA:")
{
CalSelect.VNA = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "NOISE BANDWIDTH (HZ):")
{
CalSelect.NoiseBandwidth = double.Parse(line.Split(',')[1]);
}
else if (tmpString.ToUpper() == "MT-RF100 ATTENUATION (DB):")
{
const int pathCount = 17;

for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
attenExternal.srcAttenPortModule[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}
}
else if (tmpString.ToUpper() == "MT-RF200 ATTENUATION (DB):")
{
const int pathCount = 30;

for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
attenExternal.measAttenPortModule[measIndex] = double.Parse(line.Split(',')[measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG2" || tmpString.ToUpper() ==
"NOISE")
{
CalDataPortModule caldata = new CalDataPortModule();

tmpString = line.Split(',')[0];
if (tmpString.ToUpper() == "SG1") caldata.srcSelect = SigGen.SG1;
else if (tmpString.ToUpper() == "SG2") caldata.srcSelect = SigGen.SG2;
else if (tmpString.ToUpper() == "NOISE") caldata.srcSelect = SigGen.NOISE;

tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") caldata.srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") caldata.srcPort = SrcPort.P2;
else if (tmpString.ToUpper() == "P3") caldata.srcPort = SrcPort.P3;
```

```
else if (tmpString.ToUpper() == "P4") caldata.srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") caldata.srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") caldata.srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") caldata.srcPort = SrcPort.P7;
else if (tmpString.ToUpper() == "P8") caldata.srcPort = SrcPort.P8;
else if (tmpString.ToUpper() == "P9") caldata.srcPort = SrcPort.P9;
else if (tmpString.ToUpper() == "P10") caldata.srcPort = SrcPort.P10;
else if (tmpString.ToUpper() == "P11") caldata.srcPort = SrcPort.P11;
else if (tmpString.ToUpper() == "P12") caldata.srcPort = SrcPort.P12;
else if (tmpString.ToUpper() == "P13") caldata.srcPort = SrcPort.P13;
else if (tmpString.ToUpper() == "P14") caldata.srcPort = SrcPort.P14;
else if (tmpString.ToUpper() == "P15") caldata.srcPort = SrcPort.P15;
else if (tmpString.ToUpper() == "P16") caldata.srcPort = SrcPort.P16;
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG1_OUT") caldata.srcPort =
SrcPort.SG1_OUT;
else if (tmpString.ToUpper() == "CAL") caldata.srcPort = SrcPort.CAL;
else caldata.srcPort = SrcPort.NONE;

caldata.srcPortAlias = line.Split(',')[2];

tmpString = line.Split(',')[3];
if (tmpString.ToUpper() == "P1") caldata.measPort = MeasPort.P1;
else if (tmpString.ToUpper() == "P2") caldata.measPort = MeasPort.P2;
else if (tmpString.ToUpper() == "P3") caldata.measPort = MeasPort.P3;
else if (tmpString.ToUpper() == "P4") caldata.measPort = MeasPort.P4;
else if (tmpString.ToUpper() == "P5") caldata.measPort = MeasPort.P5;
else if (tmpString.ToUpper() == "P6") caldata.measPort = MeasPort.P6;
else if (tmpString.ToUpper() == "P7") caldata.measPort = MeasPort.P7;
else if (tmpString.ToUpper() == "P8") caldata.measPort = MeasPort.P8;
else if (tmpString.ToUpper() == "P9") caldata.measPort = MeasPort.P9;
else if (tmpString.ToUpper() == "P10") caldata.measPort = MeasPort.P10;
else if (tmpString.ToUpper() == "P11") caldata.measPort = MeasPort.P11;
else if (tmpString.ToUpper() == "P12") caldata.measPort = MeasPort.P12;
else if (tmpString.ToUpper() == "P13") caldata.measPort = MeasPort.P13;
else if (tmpString.ToUpper() == "P14") caldata.measPort = MeasPort.P14;
else if (tmpString.ToUpper() == "P15") caldata.measPort = MeasPort.P15;
else if (tmpString.ToUpper() == "P16") caldata.measPort = MeasPort.P16;
else if (tmpString.ToUpper() == "P1_FWD") caldata.measPort = MeasPort.P1_Fwd;
else if (tmpString.ToUpper() == "P2_FWD") caldata.measPort = MeasPort.P2_Fwd;
else if (tmpString.ToUpper() == "P3_FWD") caldata.measPort = MeasPort.P3_Fwd;
else if (tmpString.ToUpper() == "P4_FWD") caldata.measPort = MeasPort.P4_Fwd;
else if (tmpString.ToUpper() == "P5_FWD") caldata.measPort = MeasPort.P5_Fwd;
```

```
else if (tmpString.ToUpper() == "P6_FWD") caldata.measPort = MeasPort.P6_Fwd;
else if (tmpString.ToUpper() == "P7_FWD") caldata.measPort = MeasPort.P7_Fwd;
else if (tmpString.ToUpper() == "P8_FWD") caldata.measPort = MeasPort.P8_Fwd;
else if (tmpString.ToUpper() == "P1_REV") caldata.measPort = MeasPort.P1_Rev;
else if (tmpString.ToUpper() == "P2_REV") caldata.measPort = MeasPort.P2_Rev;
else if (tmpString.ToUpper() == "P3_REV") caldata.measPort = MeasPort.P3_Rev;
else if (tmpString.ToUpper() == "P4_REV") caldata.measPort = MeasPort.P4_Rev;
else if (tmpString.ToUpper() == "P5_REV") caldata.measPort = MeasPort.P5_Rev;
else if (tmpString.ToUpper() == "P6_REV") caldata.measPort = MeasPort.P6_Rev;
else if (tmpString.ToUpper() == "P7_REV") caldata.measPort = MeasPort.P7_Rev;
else if (tmpString.ToUpper() == "P8_REV") caldata.measPort = MeasPort.P8_Rev;
else if (tmpString.ToUpper() == "M1") caldata.measPort = MeasPort.M1;
else if (tmpString.ToUpper() == "M2") caldata.measPort = MeasPort.M2;
else if (tmpString.ToUpper() == "M3") caldata.measPort = MeasPort.M3;
else if (tmpString.ToUpper() == "M4") caldata.measPort = MeasPort.M4;
else if (tmpString.ToUpper() == "M5") caldata.measPort = MeasPort.M5;
else if (tmpString.ToUpper() == "M6") caldata.measPort = MeasPort.M6;
else if (tmpString.ToUpper() == "M7") caldata.measPort = MeasPort.M7;
else if (tmpString.ToUpper() == "M8") caldata.measPort = MeasPort.M8;
else if (tmpString.ToUpper() == "M9") caldata.measPort = MeasPort.M9;
else if (tmpString.ToUpper() == "M10") caldata.measPort = MeasPort.M10;
else if (tmpString.ToUpper() == "M11") caldata.measPort = MeasPort.M11;
else if (tmpString.ToUpper() == "M12") caldata.measPort = MeasPort.M12;
else if (tmpString.ToUpper() == "M13") caldata.measPort = MeasPort.M13;
else if (tmpString.ToUpper() == "M14") caldata.measPort = MeasPort.M14;
else if (tmpString.ToUpper() == "M15") caldata.measPort = MeasPort.M15;
else if (tmpString.ToUpper() == "M16") caldata.measPort = MeasPort.M16;
else if (tmpString.ToUpper() == "M17") caldata.measPort = MeasPort.M17;
else if (tmpString.ToUpper() == "M18") caldata.measPort = MeasPort.M18;
else if (tmpString.ToUpper() == "M19") caldata.measPort = MeasPort.M19;
else if (tmpString.ToUpper() == "M20") caldata.measPort = MeasPort.M20;
else if (tmpString.ToUpper() == "M21") caldata.measPort = MeasPort.M21;
else if (tmpString.ToUpper() == "M22") caldata.measPort = MeasPort.M22;
else if (tmpString.ToUpper() == "M23") caldata.measPort = MeasPort.M23;
else if (tmpString.ToUpper() == "M24") caldata.measPort = MeasPort.M24;
else if (tmpString.ToUpper() == "M25") caldata.measPort = MeasPort.M25;
else if (tmpString.ToUpper() == "M26") caldata.measPort = MeasPort.M26;
else if (tmpString.ToUpper() == "MA") caldata.measPort = MeasPort.MA;
else if (tmpString.ToUpper() == "MB") caldata.measPort = MeasPort.MB;
else if (tmpString.ToUpper() == "MC") caldata.measPort = MeasPort.MC;
else if (tmpString.ToUpper() == "MD") caldata.measPort = MeasPort.MD;
else if (tmpString.ToUpper() == "CAL") caldata.measPort = MeasPort.CAL;
```

```
else if (tmpString.ToUpper() == "NONE") caldata.measPort = MeasPort.NONE;
else caldata.measPort = MeasPort.NONE;

caldata.measPortAlias = line.Split(',')[4];
caldata.srcFreq = double.Parse(line.Split(',')[5]);
caldata.srcLevel = double.Parse(line.Split(',')[6]);

tmpString = line.Split(',')[7];
if (tmpString.ToUpper() == "BYPASS") caldata.measFilter = MeasFilt.Bypass;
else if (tmpString.ToUpper() == "ATTEN") caldata.measFilter = MeasFilt.Atten;
else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1") caldata.measFilter =
MeasFilt.FILT1;
else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2") caldata.measFilter =
MeasFilt.FILT2;
else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3") caldata.measFilter =
MeasFilt.FILT3;
else caldata.measFilter = MeasFilt.Bypass; // Default

caldata.measAtten = double.Parse(line.Split(',')[8]);
caldata.modulationType = line.Split(',')[9];
caldata.modulationFile = line.Split(',')[10];
caldata.dutyCycle = double.Parse(line.Split(',')[11]);
caldata.comment = line.Split(',')[12];

calDatum.Add(caldata);
}
else if (tmpString.ToUpper() == "VNA-1P" || tmpString.ToUpper() == "VNA-2P")
{
// VNA calibration configuration 1

CalDataPortModule caldata1 = new CalDataPortModule();

caldata1.srcSelect = SigGen.SG1;

tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") caldata1.srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") caldata1.srcPort = SrcPort.P2;
else if (tmpString.ToUpper() == "P3") caldata1.srcPort = SrcPort.P3;
else if (tmpString.ToUpper() == "P4") caldata1.srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") caldata1.srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") caldata1.srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") caldata1.srcPort = SrcPort.P7;
```

```
else if (tmpString.ToUpper() == "P8") caldata1.srcPort = SrcPort.P8;
else caldata1.srcPort = SrcPort.NONE;

if (caldata1.srcPort == SrcPort.P1) caldata1.measPort = MeasPort.P1_Fwd;
else if (caldata1.srcPort == SrcPort.P2) caldata1.measPort = MeasPort.P2_Fwd;
else if (caldata1.srcPort == SrcPort.P3) caldata1.measPort = MeasPort.P3_Fwd;
else if (caldata1.srcPort == SrcPort.P4) caldata1.measPort = MeasPort.P4_Fwd;
else if (caldata1.srcPort == SrcPort.P5) caldata1.measPort = MeasPort.P5_Fwd;
else if (caldata1.srcPort == SrcPort.P6) caldata1.measPort = MeasPort.P6_Fwd;
else if (caldata1.srcPort == SrcPort.P7) caldata1.measPort = MeasPort.P7_Fwd;
else if (caldata1.srcPort == SrcPort.P8) caldata1.measPort = MeasPort.P8_Fwd;

caldata1.srcPortAlias = line.Split(',')[2];
caldata1.measPortAlias = line.Split(',')[4];

caldata1.srcFreq = double.Parse(line.Split(',')[5]);
caldata1.srcLevel = double.Parse(line.Split(',')[6]);

tmpString = line.Split(',')[7];
if (tmpString.ToUpper() == "BYPASS") caldata1.measFilter = MeasFilt.Bypass;
else if (tmpString.ToUpper() == "ATTEN") caldata1.measFilter = MeasFilt.Atten;
else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1") caldata1.measFilter =
MeasFilt.FILT1;
else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2") caldata1.measFilter =
MeasFilt.FILT2;
else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3") caldata1.measFilter =
MeasFilt.FILT3;
else caldata1.measFilter = MeasFilt.Bypass; // Default

caldata1.measAtten = double.Parse(line.Split(',')[8]);
caldata1.modulationType = line.Split(',')[9];
caldata1.modulationFile = line.Split(',')[10];
caldata1.dutyCycle = double.Parse(line.Split(',')[11]);
caldata1.comment = line.Split(',')[12];

calDatum.Add(caldata1);

tmpString = line.Split(',')[0];
if (tmpString.ToUpper() == "VNA-2P")
{
// VNA calibration configuration 2
```

```
CalDataPortModule caldata2 = new CalDataPortModule();

caldata2.srcSelect = SigGen.SG1;

tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") caldata2.srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") caldata2.srcPort = SrcPort.P2;
else if (tmpString.ToUpper() == "P3") caldata2.srcPort = SrcPort.P3;
else if (tmpString.ToUpper() == "P4") caldata2.srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") caldata2.srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") caldata2.srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") caldata2.srcPort = SrcPort.P7;
else if (tmpString.ToUpper() == "P8") caldata2.srcPort = SrcPort.P8;
else caldata2.srcPort = SrcPort.NONE;

tmpString = line.Split(',')[3];
if (tmpString.ToUpper() == "P1") caldata2.measPort = MeasPort.P1_Rev;
else if (tmpString.ToUpper() == "P2") caldata2.measPort = MeasPort.P2_Rev;
else if (tmpString.ToUpper() == "P3") caldata2.measPort = MeasPort.P3_Rev;
else if (tmpString.ToUpper() == "P4") caldata2.measPort = MeasPort.P4_Rev;
else if (tmpString.ToUpper() == "P5") caldata2.measPort = MeasPort.P5_Rev;
else if (tmpString.ToUpper() == "P6") caldata2.measPort = MeasPort.P6_Rev;
else if (tmpString.ToUpper() == "P7") caldata2.measPort = MeasPort.P7_Rev;
else if (tmpString.ToUpper() == "P8") caldata2.measPort = MeasPort.P8_Rev;

caldata2.srcPortAlias = line.Split(',')[2];
caldata2.measPortAlias = line.Split(',')[4];

caldata2.srcFreq = double.Parse(line.Split(',')[5]);
caldata2.srcLevel = double.Parse(line.Split(',')[6]);

tmpString = line.Split(',')[7];
if (tmpString.ToUpper() == "BYPASS") caldata2.measFilter = MeasFilt.Bypass;
else if (tmpString.ToUpper() == "ATTEN") caldata2.measFilter = MeasFilt.Atten;
else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1") caldata2.measFilter =
MeasFilt.FILT1;
else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2") caldata2.measFilter =
MeasFilt.FILT2;
else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3") caldata2.measFilter =
MeasFilt.FILT3;
else caldata2.measFilter = MeasFilt.Bypass; // Default
```

```
caldata2.measAtten = double.Parse(line.Split(',')[8]);
caldata2.modulationType = line.Split(',')[9];
caldata2.modulationFile = line.Split(',')[10];
caldata2.dutyCycle = double.Parse(line.Split(',')[11]);
caldata2.comment = line.Split(',')[12];

calDatum.Add(caldata2);

// VNA calibration configuration 3

CalDataPortModule caldata3 = new CalDataPortModule();

caldata3.srcSelect = SigGen.SG1;

tmpString = line.Split(',')[3];
if (tmpString.ToUpper() == "P1") caldata3.srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") caldata3.srcPort = SrcPort.P2;
else if (tmpString.ToUpper() == "P3") caldata3.srcPort = SrcPort.P3;
else if (tmpString.ToUpper() == "P4") caldata3.srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") caldata3.srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") caldata3.srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") caldata3.srcPort = SrcPort.P7;
else if (tmpString.ToUpper() == "P8") caldata3.srcPort = SrcPort.P8;
else caldata3.srcPort = SrcPort.NONE;

if (caldata3.srcPort == SrcPort.P1) caldata3.measPort = MeasPort.P1_Fwd;
else if (caldata3.srcPort == SrcPort.P2) caldata3.measPort = MeasPort.P2_Fwd;
else if (caldata3.srcPort == SrcPort.P3) caldata3.measPort = MeasPort.P3_Fwd;
else if (caldata3.srcPort == SrcPort.P4) caldata3.measPort = MeasPort.P4_Fwd;
else if (caldata3.srcPort == SrcPort.P5) caldata3.measPort = MeasPort.P5_Fwd;
else if (caldata3.srcPort == SrcPort.P6) caldata3.measPort = MeasPort.P6_Fwd;
else if (caldata3.srcPort == SrcPort.P7) caldata3.measPort = MeasPort.P7_Fwd;
else if (caldata3.srcPort == SrcPort.P8) caldata3.measPort = MeasPort.P8_Fwd;

caldata3.srcPortAlias = line.Split(',')[2];
caldata3.measPortAlias = line.Split(',')[4];

caldata3.srcFreq = double.Parse(line.Split(',')[5]);
caldata3.srcLevel = double.Parse(line.Split(',')[6]);

tmpString = line.Split(',')[7];
if (tmpString.ToUpper() == "BYPASS") caldata3.measFilter = MeasFilt.Bypass;
```

```
else if (tmpString.ToUpper() == "ATTEN") caldata3.measFilter = MeasFilt.Atten;
else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1") caldata3.measFilter =
MeasFilt.FILT1;
else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2") caldata3.measFilter =
MeasFilt.FILT2;
else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3") caldata3.measFilter =
MeasFilt.FILT3;
else caldata3.measFilter = MeasFilt.Bypass; // Default

caldata3.measAtten = double.Parse(line.Split(',')[8]);
caldata3.modulationType = line.Split(',')[9];
caldata3.modulationFile = line.Split(',')[10];
caldata3.dutyCycle = double.Parse(line.Split(',')[11]);
caldata3.comment = line.Split(',')[12];

calDatum.Add(caldata3);

// VNA calibration configuration 4

CalDataPortModule caldata4 = new CalDataPortModule();

caldata4.srcSelect = SigGen.SG1;

tmpString = line.Split(',')[3];
if (tmpString.ToUpper() == "P1") caldata4.srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") caldata4.srcPort = SrcPort.P2;
else if (tmpString.ToUpper() == "P3") caldata4.srcPort = SrcPort.P3;
else if (tmpString.ToUpper() == "P4") caldata4.srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") caldata4.srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") caldata4.srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") caldata4.srcPort = SrcPort.P7;
else if (tmpString.ToUpper() == "P8") caldata4.srcPort = SrcPort.P8;
else caldata4.srcPort = SrcPort.NONE;

tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") caldata4.measPort = MeasPort.P1_Rev;
else if (tmpString.ToUpper() == "P2") caldata4.measPort = MeasPort.P2_Rev;
else if (tmpString.ToUpper() == "P3") caldata4.measPort = MeasPort.P3_Rev;
else if (tmpString.ToUpper() == "P4") caldata4.measPort = MeasPort.P4_Rev;
else if (tmpString.ToUpper() == "P5") caldata4.measPort = MeasPort.P5_Rev;
else if (tmpString.ToUpper() == "P6") caldata4.measPort = MeasPort.P6_Rev;
else if (tmpString.ToUpper() == "P7") caldata4.measPort = MeasPort.P7_Rev;
```

```
else if (tmpString.ToUpper() == "P8") caldata4.measPort = MeasPort.P8_Rev;

caldata4.srcPortAlias = line.Split(',')[2];
caldata4.measPortAlias = line.Split(',')[4];

caldata4.srcFreq = double.Parse(line.Split(',')[5]);
caldata4.srcLevel = double.Parse(line.Split(',')[6]);

tmpString = line.Split(',')[7];
if (tmpString.ToUpper() == "BYPASS") caldata4.measFilter = MeasFilt.Bypass;
else if (tmpString.ToUpper() == "ATTEN") caldata4.measFilter = MeasFilt.Atten;
else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1") caldata4.measFilter =
MeasFilt.FILT1;
else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2") caldata4.measFilter =
MeasFilt.FILT2;
else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3") caldata4.measFilter =
MeasFilt.FILT3;
else caldata4.measFilter = MeasFilt.Bypass; // Default

caldata4.measAtten = double.Parse(line.Split(',')[8]);
caldata4.modulationType = line.Split(',')[9];
caldata4.modulationFile = line.Split(',')[10];
caldata4.dutyCycle = double.Parse(line.Split(',')[11]);
caldata4.comment = line.Split(',')[12];

calDatum.Add(caldata4);
}
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
}

return Tuple.Create(info, CalSelect, attenExternal, calDatum);
}

/// <summary>
/// Converts a string into a boolean.
/// </summary>
```

```
/// <param name="Value"></param>
/// <returns></returns>
private static bool ConvertStringToBool(string Value, string ErrorText, int LineNumber)
{

Value = Value.Trim().ToLower();

switch (Value)
{
case "yes":
case "y":
case "true":
case "pass":
case "p":
case "1":
return true;
case "false":
case "fail":
case "0":
case "f":
case "n":
case "no":
return false;
default:
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. (Use
Y or N) ", ErrorText, LineNumber, Value));
}
}

/// <summary>
/// Converts a string into a double.
/// </summary>
/// <param name="Value"></param>
/// <returns>A double representing the string or 0 if the string can't be converted.</returns>
private static double ConvertStringToDouble(string Value, string ErrorText, int LineNumber)
{
if (double.TryParse(Value, out double result) == true)
{
return result;
}
else
{
```

```
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
ErrorText, LineNumber, Value));
}
}

/// <summary>
/// Converts a string into a short.
/// </summary>
/// <param name="Value"></param>
/// <returns>A short representing the string.</returns>
/// <exception cref="CalibrationDefinitionException"></exception>
private static short ConvertStringToShort(string Value, string ErrorText, int LineNumber)
{
if (short.TryParse(Value, out short result) == true)
{
return result;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
ErrorText, LineNumber, Value));
}
}

/// <summary>
/// Reads in a Merlin Test Version 4 User Calibration Definition file.
/// </summary>
/// <param name="calConfigFile">The filename of the calibration definition file.</param>
/// <param name="Expand">True if the list of VNA Cal setting VNA-1P and VNA-2P are to be
expanded or not.</param>
/// <returns>A tuple object consisting of four objects representing the data read from the specified
calibration definition file.</returns>
public Tuple<Info, CalSettings, ExternalAttenuation, List<CalDataPortModule>>
ImportCalConfigVer4V2(string calConfigFile, bool Expand = true)
{
int numberOfRowRequiredToGetHeaderInfo = 20;
int linePortDataStartsAt = int.MinValue;

Info info = new Info();
CalSettings CalSelect = new CalSettings();
ExternalAttenuation attenExternal = new ExternalAttenuation();
List<CalDataPortModule> calDatum = new List<CalDataPortModule>();
```

```
// Read header of file and split by ','
var headerInfo = (from lines in
File.ReadLines(calConfigFile).Take(numberOfRowRequiredToGetHeaderInfo)
let columns = lines.Split(',')
select columns).ToList();

// Meta Data Stored in Header
for (int index = 0; index < headerInfo.Count; index++)
{
if (headerInfo[index].Length > 1)
{
// Product Name
if (headerInfo[index][0].ToUpper() == "PRODUCT:")
{
info.Product = headerInfo[index][1];
}

// Revision
if (headerInfo[index][0].ToUpper() == "REVISION:")
{
info.Revision = headerInfo[index][1];
}

// Test Program
if (headerInfo[index][0].ToUpper() == "TEST PROGRAM:")
{
info.Program = headerInfo[index][1];
}

// Calibration Version.
if (headerInfo[index][0].ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = headerInfo[index][1];
}

// Comment
if (headerInfo[index][0].ToUpper() == "COMMENT:")
{
info.Comment = headerInfo[index][1];
}
```

```
if (headerInfo[index][0].ToUpper() == "CALIBRATE RF SOURCE/MEASURE:")
{
CalSelect.SourceMeasurePath = ConvertStringToBool(headerInfo[index][1], headerInfo[index][0],
index);
}

if (headerInfo[index][0].ToUpper() == "CALIBRATE NOISE SOURCE:")
{
CalSelect.NoiseSource = ConvertStringToBool(headerInfo[index][1], headerInfo[index][0], index);
}

if (headerInfo[index][0].ToUpper() == "CALIBRATE VNA:")
{
CalSelect.VNA = ConvertStringToBool(headerInfo[index][1], headerInfo[index][0], index);
}

if (headerInfo[index][0].ToUpper() == "NOISE BANDWIDTH (HZ):")
{
CalSelect.NoiseBandwidth = ConvertStringToDouble(headerInfo[index][1], headerInfo[index][0],
index);
}

if (headerInfo[index][0].ToUpper() == "MT-RF100 ATTENUATION (DB):")
{
const int pathCount = 17;

if (headerInfo[index].Length > pathCount)
{
for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
// If an Error occurs to help the user identify where the error is we mark the actual element we're
on in the errorText.
string errorText = string.Format("{0}: Element {1}", headerInfo[index][0], srcIndex + 1);
attenExternal.srcAttenPortModule[srcIndex] = ConvertStringToDouble(headerInfo[index][srcIndex
+ 1], errorText, index);
}
}
else
{

throw new CalibrationDefinitionException(string.Format("MT-RF100 ATTENUATION (DB): does
not have {0} elements.", pathCount));
```

```
}
}

if (headerInfo[index][0].ToUpper() == "MT-RF200 ATTENUATION (DB):")
{
const int pathCount = 30;

if (headerInfo[index].Length > pathCount)
{
for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
// If an Error occurs to help the user identify where the error is we mark the actual element we're
on in the errorText.
string errorText = string.Format("{0}: Element {1}", headerInfo[index][0], measIndex + 1);
attenExternal.measAttenPortModule[measIndex] =
ConvertStringToDouble(headerInfo[index][measIndex + 1], errorText, index);
}
}
else
{
throw new CalibrationDefinitionException(string.Format("MT-RF200 ATTENUATION (DB): does
not have {0} elements.", pathCount));
}
}

if (headerInfo[index][0].ToUpper() == "SOURCE")
{
linePortDataStartsAt = index;
}
}
}

// Now Read in Port Settings
// Read header of file and split by ','
var portSettings = (from lines in File.ReadLines(calConfigFile).Skip(linePortDataStartsAt)
let columns = lines.Split(',')
select columns).ToList();

// Now check to make sure values were read in for Product, and CAL Version.

for (int index = 1; index < portSettings.Count; index++)
{
```

```
SigGen sigGen;
SrcPort srcPort;
MeasPort measPort;
MeasFilt measFilt;
bool vnaSrc = false;

CalDataPortModule caldata = new CalDataPortModule();

if (Expand == true)
{
// Source
if (Enum.TryParse<SigGen>(portSettings[index][0], true, out sigGen) == true)
{
caldata.srcSelect = sigGen;
}
else
{
// Handle VNA Cal Case
if (portSettings[index][0].ToUpper() == "VNA-1P" || portSettings[index][0].ToUpper() == "VNA-2P")
{
caldata.srcSelect = SigGen.SG1;
vnaSrc = true;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][0], index + linePortDataStartsAt + 1, portSettings[index][0]));
}
}
}
else
{
// Source
if (Enum.TryParse<SigGen>(portSettings[index][0], true, out sigGen) == true)
{
caldata.srcSelect = sigGen;
}
else
{
// Handle VNA Cal Case
if (portSettings[index][0].ToUpper() == "VNA-1P")
{
```

```
caldata.srcSelect = SigGen.VNA_1P;
vnaSrc = true;
}
else if (portSettings[index][0].ToUpper() == "VNA-2P")
{
caldata.srcSelect = SigGen.VNA_2P;
vnaSrc = true;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][0], index + linePortDataStartsAt + 1, portSettings[index][0]));
}
}
}

if (vnaSrc == true && Expand == true)
{
// VNA calibration configuration 1
CalDataPortModule caldata1 = new CalDataPortModule();

caldata1.srcSelect = SigGen.SG1;

// Source Port
if (Enum.TryParse<SrcPort>(portSettings[index][1], true, out srcPort) == true)
{
caldata1.srcPort = srcPort;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][1], index + linePortDataStartsAt + 1, portSettings[index][1]));
}

// Source Port Alias
caldata1.srcPortAlias = portSettings[index][2];



if (caldata1.srcPort == SrcPort.P1) caldata1.measPort = MeasPort.P1_Fwd;
else if (caldata1.srcPort == SrcPort.P2) caldata1.measPort = MeasPort.P2_Fwd;
else if (caldata1.srcPort == SrcPort.P3) caldata1.measPort = MeasPort.P3_Fwd;
else if (caldata1.srcPort == SrcPort.P4) caldata1.measPort = MeasPort.P4_Fwd;
```

```
else if (caldata1.srcPort == SrcPort.P5) caldata1.measPort = MeasPort.P5_Fwd;
else if (caldata1.srcPort == SrcPort.P6) caldata1.measPort = MeasPort.P6_Fwd;
else if (caldata1.srcPort == SrcPort.P7) caldata1.measPort = MeasPort.P7_Fwd;
else if (caldata1.srcPort == SrcPort.P8) caldata1.measPort = MeasPort.P8_Fwd;
else throw new CalibrationDefinitionException(string.Format("Source Port Value Invalid : {0} - Line
{1}. (Valid values of P1 to P8.)", caldata1.srcPort, index + linePortDataStartsAt + 1));

// Measure Port Alias
caldata1.measPortAlias = portSettings[index][4];

// Frequency
caldata1.srcFreq = ConvertStringToDouble(portSettings[index][5], portSettings[0][5], index +
linePortDataStartsAt + 1);

// Level
caldata1.srcLevel = ConvertStringToDouble(portSettings[index][6], portSettings[0][6], index +
linePortDataStartsAt + 1);

// Filter
if (Enum.TryParse<MeasFilt>(portSettings[index][7], true, out measFilt) == true)
{
caldata1.measFilter = measFilt;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][7], index + linePortDataStartsAt + 1, portSettings[index][7]));
}

// Meas Attenuation
caldata1.measAtten = ConvertStringToDouble(portSettings[index][8], portSettings[0][8], index +
linePortDataStartsAt + 1);

// Modulation Type
caldata1.modulationType = portSettings[index][9];

// Modulation File
caldata1.modulationFile = portSettings[index][10];

// Duty Cycle
caldata1.dutyCycle = ConvertStringToDouble(portSettings[index][11], portSettings[0][11], index +
linePortDataStartsAt + 1);
```

```
// Comment
caldata1.comment = portSettings[index][12];

calDatum.Add(caldata1);



if (portSettings[index][0].ToUpper() == "VNA-2P")
{
// VNA calibration configuration 2

CalDataPortModule caldata2 = new CalDataPortModule();

caldata2.srcSelect = SigGen.SG1;

// Source Port
if (Enum.TryParse<SrcPort>(portSettings[index][1], true, out srcPort) == true)
{
caldata2.srcPort = srcPort;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][1], index + linePortDataStartsAt + 1, portSettings[index][1]));
}

// Source Port Alias
caldata2.srcPortAlias = portSettings[index][2];

if (caldata2.srcPort == SrcPort.P1) caldata2.measPort = MeasPort.P1_Rev;
else if (caldata2.srcPort == SrcPort.P2) caldata2.measPort = MeasPort.P2_Rev;
else if (caldata2.srcPort == SrcPort.P3) caldata2.measPort = MeasPort.P3_Rev;
else if (caldata2.srcPort == SrcPort.P4) caldata2.measPort = MeasPort.P4_Rev;
else if (caldata2.srcPort == SrcPort.P5) caldata2.measPort = MeasPort.P5_Rev;
else if (caldata2.srcPort == SrcPort.P6) caldata2.measPort = MeasPort.P6_Rev;
else if (caldata2.srcPort == SrcPort.P7) caldata2.measPort = MeasPort.P7_Rev;
else if (caldata2.srcPort == SrcPort.P8) caldata2.measPort = MeasPort.P8_Rev;
else throw new CalibrationDefinitionException(string.Format("Source Port Value Invalid : {0} - Line
{1}. (Valid values of P1 to P8.)", caldata2.srcPort, index + linePortDataStartsAt + 1));

// Measure Port Alias
caldata2.measPortAlias = portSettings[index][4];
```

```
// Frequency
caldata2.srcFreq = ConvertStringToDouble(portSettings[index][5], portSettings[0][5], index +
linePortDataStartsAt + 1);

// Level
caldata2.srcLevel = ConvertStringToDouble(portSettings[index][6], portSettings[0][6], index +
linePortDataStartsAt + 1);

// Filter
if (Enum.TryParse<MeasFilt>(portSettings[index][7], true, out measFilt) == true)
{
caldata2.measFilter = measFilt;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][7], index + linePortDataStartsAt + 1, portSettings[index][7]));
}

// Meas Attenuation
caldata2.measAtten = ConvertStringToDouble(portSettings[index][8], portSettings[0][8], index +
linePortDataStartsAt + 1);

// Modulation Type
caldata2.modulationType = portSettings[index][9];

// Modulation File
caldata2.modulationFile = portSettings[index][10];

// Duty Cycle
caldata2.dutyCycle = ConvertStringToDouble(portSettings[index][11], portSettings[0][11], index +
linePortDataStartsAt + 1);

// Comment
caldata2.comment = portSettings[index][12];

calDatum.Add(caldata2);

// VNA calibration configuration 3

CalDataPortModule caldata3 = new CalDataPortModule();
```

```
caldata3.srcSelect = SigGen.SG1;

// Source Port
if (Enum.TryParse<SrcPort>(portSettings[index][1], true, out srcPort) == true)
{
caldata3.srcPort = srcPort;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][1], index + linePortDataStartsAt + 1, portSettings[index][1]));
}

// Source Port Alias
caldata3.srcPortAlias = portSettings[index][2];

if (caldata3.srcPort == SrcPort.P1) caldata3.measPort = MeasPort.P1_Fwd;
else if (caldata3.srcPort == SrcPort.P2) caldata3.measPort = MeasPort.P2_Fwd;
else if (caldata3.srcPort == SrcPort.P3) caldata3.measPort = MeasPort.P3_Fwd;
else if (caldata3.srcPort == SrcPort.P4) caldata3.measPort = MeasPort.P4_Fwd;
else if (caldata3.srcPort == SrcPort.P5) caldata3.measPort = MeasPort.P5_Fwd;
else if (caldata3.srcPort == SrcPort.P6) caldata3.measPort = MeasPort.P6_Fwd;
else if (caldata3.srcPort == SrcPort.P7) caldata3.measPort = MeasPort.P7_Fwd;
else if (caldata3.srcPort == SrcPort.P8) caldata3.measPort = MeasPort.P8_Fwd;
else throw new CalibrationDefinitionException(string.Format("Source Port Value Invalid : {0} - Line
{1}. (Valid values of P1 to P8.)", caldata3.srcPort, index + linePortDataStartsAt + 1));

// Measure Port Alias
caldata3.measPortAlias = portSettings[index][4];

// Frequency
caldata3.srcFreq = ConvertStringToDouble(portSettings[index][5], portSettings[0][5], index +
linePortDataStartsAt + 1);

// Level
caldata3.srcLevel = ConvertStringToDouble(portSettings[index][6], portSettings[0][6], index +
linePortDataStartsAt + 1);

// Filter
if (Enum.TryParse<MeasFilt>(portSettings[index][7], true, out measFilt) == true)
{
```

```
caldata3.measFilter = measFilt;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][7], index + linePortDataStartsAt + 1, portSettings[index][7]));
}

// Meas Attenuation
caldata3.measAtten = ConvertStringToDouble(portSettings[index][8], portSettings[0][8], index +
linePortDataStartsAt + 1);

// Modulation Type
caldata3.modulationType = portSettings[index][9];

// Modulation File
caldata3.modulationFile = portSettings[index][10];

// Duty Cycle
caldata3.dutyCycle = ConvertStringToDouble(portSettings[index][11], portSettings[0][11], index +
linePortDataStartsAt + 1);

// Comment
caldata3.comment = portSettings[index][12];

calDatum.Add(caldata3);

// VNA calibration configuration 4

CalDataPortModule caldata4 = new CalDataPortModule();

caldata4.srcSelect = SigGen.SG1;

// Source Port
if (Enum.TryParse<SrcPort>(portSettings[index][1], true, out srcPort) == true)
{
caldata4.srcPort = srcPort;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][1], index + linePortDataStartsAt + 1, portSettings[index][1]));
```

```
}

// Source Port Alias
caldata4.srcPortAlias = portSettings[index][2];

if (caldata4.srcPort == SrcPort.P1) caldata4.measPort = MeasPort.P1_Rev;
else if (caldata4.srcPort == SrcPort.P2) caldata4.measPort = MeasPort.P2_Rev;
else if (caldata4.srcPort == SrcPort.P3) caldata4.measPort = MeasPort.P3_Rev;
else if (caldata4.srcPort == SrcPort.P4) caldata4.measPort = MeasPort.P4_Rev;
else if (caldata4.srcPort == SrcPort.P5) caldata4.measPort = MeasPort.P5_Rev;
else if (caldata4.srcPort == SrcPort.P6) caldata4.measPort = MeasPort.P6_Rev;
else if (caldata4.srcPort == SrcPort.P7) caldata4.measPort = MeasPort.P7_Rev;
else if (caldata4.srcPort == SrcPort.P8) caldata4.measPort = MeasPort.P8_Rev;
else throw new CalibrationDefinitionException(string.Format("Source Port Value Invalid : {0} - Line
{1}. (Valid values of P1 to P8.)", caldata4.srcPort, index + linePortDataStartsAt + 1));

// Measure Port Alias
caldata4.measPortAlias = portSettings[index][4];

// Frequency
caldata4.srcFreq = ConvertStringToDouble(portSettings[index][5], portSettings[0][5], index +
linePortDataStartsAt + 1);

// Level
caldata4.srcLevel = ConvertStringToDouble(portSettings[index][6], portSettings[0][6], index +
linePortDataStartsAt + 1);

// Filter
if (Enum.TryParse<MeasFilt>(portSettings[index][7], true, out measFilt) == true)
{
caldata4.measFilter = measFilt;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][7], index + linePortDataStartsAt + 1, portSettings[index][7]));
}

// Meas Attenuation
caldata4.measAtten = ConvertStringToDouble(portSettings[index][8], portSettings[0][8], index +
linePortDataStartsAt + 1);
```

```
// Modulation Type
caldata4.modulationType = portSettings[index][9];

// Modulation File
caldata4.modulationFile = portSettings[index][10];

// Duty Cycle
caldata4.dutyCycle = ConvertStringToDouble(portSettings[index][11], portSettings[0][11], index +
linePortDataStartsAt + 1);

// Comment
caldata4.comment = portSettings[index][12];

calDatum.Add(caldata4);
}
}
else
{
// Source Port
if (Enum.TryParse<SrcPort>(portSettings[index][1], true, out srcPort) == true)
{
caldata.srcPort = srcPort;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][1], index + linePortDataStartsAt + 1, portSettings[index][1]));
}

// Source Port Alias
caldata.srcPortAlias = portSettings[index][2];

// Measure Port
if (Enum.TryParse<MeasPort>(portSettings[index][3], true, out measPort) == true)
{
caldata.measPort = measPort;
}
else
{
caldata.measPort = MeasPort.NONE;
//throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][3], index + linePortDataStartsAt + 1, portSettings[index][3]));
```

```
}


// Measure Port Alias
caldata.measPortAlias = portSettings[index][4];

// Frequency
caldata.srcFreq = ConvertStringToDouble(portSettings[index][5], portSettings[0][5], index +
linePortDataStartsAt + 1);

// Level
caldata.srcLevel = ConvertStringToDouble(portSettings[index][6], portSettings[0][6], index +
linePortDataStartsAt + 1);

// Filter
if (Enum.TryParse<MeasFilt>(portSettings[index][7], true, out measFilt) == true)
{
caldata.measFilter = measFilt;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][7], index + linePortDataStartsAt + 1, portSettings[index][7]));
}

// Meas Attenuation
caldata.measAtten = ConvertStringToDouble(portSettings[index][8], portSettings[0][8], index +
linePortDataStartsAt + 1);

// Modulation Type
caldata.modulationType = portSettings[index][9];

// Modulation File
caldata.modulationFile = portSettings[index][10];

// Duty Cycle
caldata.dutyCycle = ConvertStringToDouble(portSettings[index][11], portSettings[0][11], index +
linePortDataStartsAt + 1);

// Comment
caldata.comment = portSettings[index][12];
```

```
calDatum.Add(caldata);
}
}

if (string.IsNullOrWhiteSpace(info.Product) == true)
{
throw new CalibrationDefinitionException("Product Value not defined.");
}

if (string.IsNullOrWhiteSpace(info.CalVersion) == true)
{
throw new CalibrationDefinitionException("Calibration Version Value not defined.");
}

return Tuple.Create(info, CalSelect, attenExternal, calDatum);
}

/// <summary>
/// Reads in a Merlin Test Version 4 User Calibration Definition file.
/// </summary>
/// <param name="calConfigFile">The filename of the calibration definition file.</param>
/// <param name="Expand">True if the list of VNA Cal setting VNA-1P and VNA-2P are to be
expanded or not.</param>
/// <returns>A tuple object consisting of four objects representing the data read from the specified
calibration definition file.</returns>
public Tuple<Info, CalSettings, ExternalAttenuation, List<CalDataPortModule>>
ImportCalConfigVer5V2(string calConfigFile, bool Expand = true)
{
int numberOfRowRequiredToGetHeaderInfo = 21;
int linePortDataStartsAt = int.MinValue;

Info info = new Info();
CalSettings CalSelect = new CalSettings();
ExternalAttenuation attenExternal = new ExternalAttenuation();
List<CalDataPortModule> calDatum = new List<CalDataPortModule>();

// Read header of file and split by ','
var headerInfo = (from lines in
File.ReadLines(calConfigFile).Take(numberOfRowRequiredToGetHeaderInfo)
let columns = lines.Split(',')
select columns).ToList();
```

```
// Meta Data Stored in Header
for (int index = 0; index < headerInfo.Count; index++)
{
if (headerInfo[index].Length > 1)
{
// Product Name
if (headerInfo[index][0].ToUpper() == "PRODUCT:")
{
info.Product = headerInfo[index][1];
}

// Revision
if (headerInfo[index][0].ToUpper() == "REVISION:")
{
info.Revision = headerInfo[index][1];
}

// Test Program
if (headerInfo[index][0].ToUpper() == "TEST PROGRAM:")
{
info.Program = headerInfo[index][1];
}

// Calibration Version.
if (headerInfo[index][0].ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = headerInfo[index][1];
}

// Comment
if (headerInfo[index][0].ToUpper() == "COMMENT:")
{
info.Comment = headerInfo[index][1];
}

if (headerInfo[index][0].ToUpper() == "CALIBRATE RF SOURCE/MEASURE:")
{
CalSelect.SourceMeasurePath = ConvertStringToBool(headerInfo[index][1], headerInfo[index][0],
index);
}

if (headerInfo[index][0].ToUpper() == "CALIBRATE NOISE SOURCE:")
```

```
{
CalSelect.NoiseSource = ConvertStringToBool(headerInfo[index][1], headerInfo[index][0], index);
}

if (headerInfo[index][0].ToUpper() == "CALIBRATE VNA:")
{
CalSelect.VNA = ConvertStringToBool(headerInfo[index][1], headerInfo[index][0], index);
}

if (headerInfo[index][0].ToUpper() == "NOISE BANDWIDTH (HZ):")
{
CalSelect.NoiseBandwidth = ConvertStringToDouble(headerInfo[index][1], headerInfo[index][0],
index);
}

if (headerInfo[index][0].ToUpper() == "MT-RF100 ATTENUATION (DB):")
{
const int pathCount = 17;

if (headerInfo[index].Length > pathCount)
{
for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
{
// If an Error occurs to help the user identify where the error is we mark the actual element we're
on in the errorText.
string errorText = string.Format("{0}: Element {1}", headerInfo[index][0], srcIndex + 1);
attenExternal.srcAttenPortModule[srcIndex] = ConvertStringToDouble(headerInfo[index][srcIndex
+ 1], errorText, index);
}
}
else
{

throw new CalibrationDefinitionException(string.Format("MT-RF100 ATTENUATION (DB): does
not have {0} elements.", pathCount));
}
}

if (headerInfo[index][0].ToUpper() == "MT-RF200 ATTENUATION (DB):")
{
const int pathCount = 30;
```

```
if (headerInfo[index].Length > pathCount)
{
for (int measIndex = 0; measIndex < pathCount; measIndex++)
{
// If an Error occurs to help the user identify where the error is we mark the actual element we're
on in the errorText.
string errorText = string.Format("{0}: Element {1}", headerInfo[index][0], measIndex + 1);
attenExternal.measAttenPortModule[measIndex] =
ConvertStringToDouble(headerInfo[index][measIndex + 1], errorText, index);
}
}
else
{
throw new CalibrationDefinitionException(string.Format("MT-RF200 ATTENUATION (DB): does
not have {0} elements.", pathCount));
}
}

if (headerInfo[index][0].ToUpper() == "SOURCE")
{
linePortDataStartsAt = index;
}
}
}

// Now Read in Port Settings
// Read header of file and split by ','
var portSettings = (from lines in File.ReadLines(calConfigFile).Skip(linePortDataStartsAt)
let columns = lines.Split(',')
select columns).ToList();

// Now check to make sure values were read in for Product, and CAL Version.

for (int index = 1; index < portSettings.Count; index++)
{
SigGen sigGen;
SrcPort srcPort;
MeasPort measPort;
MeasFilt measFilt;
bool vnaSrc = false;

CalDataPortModule caldata = new CalDataPortModule();
```

```
if (Expand == true)
{
// Source
if (Enum.TryParse<SigGen>(portSettings[index][0], true, out sigGen) == true)
{
caldata.srcSelect = sigGen;
}
else
{
// Handle VNA Cal Case
if (portSettings[index][0].ToUpper() == "VNA-1P" || portSettings[index][0].ToUpper() == "VNA-2P")
{
caldata.srcSelect = SigGen.SG1;
vnaSrc = true;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][0], index + linePortDataStartsAt + 1, portSettings[index][0]));
}
}
}
else
{
// Source
if (Enum.TryParse<SigGen>(portSettings[index][0], true, out sigGen) == true)
{
caldata.srcSelect = sigGen;
}
else
{
// Handle VNA Cal Case
if (portSettings[index][0].ToUpper() == "VNA-1P")
{
caldata.srcSelect = SigGen.VNA_1P;
vnaSrc = true;
}
else if (portSettings[index][0].ToUpper() == "VNA-2P")
{
caldata.srcSelect = SigGen.VNA_2P;
vnaSrc = true;
```

```
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][0], index + linePortDataStartsAt + 1, portSettings[index][0]));
}
}
}

if (vnaSrc == true && Expand == true)
{
// VNA calibration configuration 1
CalDataPortModule caldata1 = new CalDataPortModule();

caldata1.srcSelect = SigGen.SG1;

// Source Port
if (Enum.TryParse<SrcPort>(portSettings[index][1], true, out srcPort) == true)
{
caldata1.srcPort = srcPort;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][1], index + linePortDataStartsAt + 1, portSettings[index][1]));
}

// Source Port Alias
caldata1.srcPortAlias = portSettings[index][2];


if (caldata1.srcPort == SrcPort.P1) caldata1.measPort = MeasPort.P1_Fwd;
else if (caldata1.srcPort == SrcPort.P2) caldata1.measPort = MeasPort.P2_Fwd;
else if (caldata1.srcPort == SrcPort.P3) caldata1.measPort = MeasPort.P3_Fwd;
else if (caldata1.srcPort == SrcPort.P4) caldata1.measPort = MeasPort.P4_Fwd;
else if (caldata1.srcPort == SrcPort.P5) caldata1.measPort = MeasPort.P5_Fwd;
else if (caldata1.srcPort == SrcPort.P6) caldata1.measPort = MeasPort.P6_Fwd;
else if (caldata1.srcPort == SrcPort.P7) caldata1.measPort = MeasPort.P7_Fwd;
else if (caldata1.srcPort == SrcPort.P8) caldata1.measPort = MeasPort.P8_Fwd;
else throw new CalibrationDefinitionException(string.Format("Source Port Value Invalid : {0} - Line
{1}. (Valid values of P1 to P8.)", caldata1.srcPort, index + linePortDataStartsAt + 1));
```

```
// Measure Port Alias
caldata1.measPortAlias = portSettings[index][4];

// Frequency
caldata1.srcFreq = ConvertStringToDouble(portSettings[index][5], portSettings[0][5], index +
linePortDataStartsAt + 1);

// Level
caldata1.srcLevel = ConvertStringToDouble(portSettings[index][6], portSettings[0][6], index +
linePortDataStartsAt + 1);

// Filter
if (Enum.TryParse<MeasFilt>(portSettings[index][7], true, out measFilt) == true)
{
caldata1.measFilter = measFilt;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][7], index + linePortDataStartsAt + 1, portSettings[index][7]));
}

// Meas Attenuation
caldata1.measAtten = ConvertStringToDouble(portSettings[index][8], portSettings[0][8], index +
linePortDataStartsAt + 1);

// Modulation Type
caldata1.modulationType = portSettings[index][9];

// Modulation File
caldata1.modulationFile = portSettings[index][10];

// Waveform settings
caldata1.WaveConfig = new WaveformSettings();

// Marker
if (portSettings[index][11] != string.Empty)
{
if (portSettings[index][11].ToUpper() == "EXT")
{
caldata1.WaveConfig.Marker = 0;
}
```

```
else
{
caldata1.WaveConfig.Marker = ConvertStringToShort(portSettings[index][11], portSettings[0][11],
index + linePortDataStartsAt + 1);
}
}
else
{
caldata1.WaveConfig.Marker = -1;
}

// Analysis slot
if (portSettings[index][12] != string.Empty)
{
caldata1.WaveConfig.AnalysisSlot = ConvertStringToShort(portSettings[index][12],
portSettings[0][12], index + linePortDataStartsAt + 1);
}
else
{
caldata1.WaveConfig.AnalysisSlot = -1;
}



// Comment
caldata1.comment = portSettings[index][18];

calDatum.Add(caldata1);



if (portSettings[index][0].ToUpper() == "VNA-2P")
{
// VNA calibration configuration 2

CalDataPortModule caldata2 = new CalDataPortModule();

caldata2.srcSelect = SigGen.SG1;

// Source Port
if (Enum.TryParse<SrcPort>(portSettings[index][1], true, out srcPort) == true)
{
caldata2.srcPort = srcPort;
}
```

```
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][1], index + linePortDataStartsAt + 1, portSettings[index][1]));
}

// Source Port Alias
caldata2.srcPortAlias = portSettings[index][2];

if (caldata2.srcPort == SrcPort.P1) caldata2.measPort = MeasPort.P1_Rev;
else if (caldata2.srcPort == SrcPort.P2) caldata2.measPort = MeasPort.P2_Rev;
else if (caldata2.srcPort == SrcPort.P3) caldata2.measPort = MeasPort.P3_Rev;
else if (caldata2.srcPort == SrcPort.P4) caldata2.measPort = MeasPort.P4_Rev;
else if (caldata2.srcPort == SrcPort.P5) caldata2.measPort = MeasPort.P5_Rev;
else if (caldata2.srcPort == SrcPort.P6) caldata2.measPort = MeasPort.P6_Rev;
else if (caldata2.srcPort == SrcPort.P7) caldata2.measPort = MeasPort.P7_Rev;
else if (caldata2.srcPort == SrcPort.P8) caldata2.measPort = MeasPort.P8_Rev;
else throw new CalibrationDefinitionException(string.Format("Source Port Value Invalid : {0} - Line
{1}. (Valid values of P1 to P8.)", caldata2.srcPort, index + linePortDataStartsAt + 1));

// Measure Port Alias
caldata2.measPortAlias = portSettings[index][4];

// Frequency
caldata2.srcFreq = ConvertStringToDouble(portSettings[index][5], portSettings[0][5], index +
linePortDataStartsAt + 1);

// Level
caldata2.srcLevel = ConvertStringToDouble(portSettings[index][6], portSettings[0][6], index +
linePortDataStartsAt + 1);

// Filter
if (Enum.TryParse<MeasFilt>(portSettings[index][7], true, out measFilt) == true)
{
caldata2.measFilter = measFilt;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][7], index + linePortDataStartsAt + 1, portSettings[index][7]));
}
```

```
// Meas Attenuation
caldata2.measAtten = ConvertStringToDouble(portSettings[index][8], portSettings[0][8], index +
linePortDataStartsAt + 1);

// Modulation Type
caldata2.modulationType = portSettings[index][9];

// Modulation File
caldata2.modulationFile = portSettings[index][10];



// Waveform settings
caldata2.WaveConfig = new WaveformSettings();

// Marker
if (portSettings[index][11] != string.Empty)
{
if (portSettings[index][11].ToUpper() == "EXT")
{
caldata2.WaveConfig.Marker = 0;
}
else
{
caldata2.WaveConfig.Marker = ConvertStringToShort(portSettings[index][11], portSettings[0][11],
index + linePortDataStartsAt + 1);
}
}
else
{
caldata2.WaveConfig.Marker = -1;
}

// Analysis slot
if (portSettings[index][12] != string.Empty)
{
caldata2.WaveConfig.AnalysisSlot = ConvertStringToShort(portSettings[index][12],
portSettings[0][12], index + linePortDataStartsAt + 1);
}
else
{
caldata2.WaveConfig.AnalysisSlot = -1;
}
```

```
// Comment
caldata2.comment = portSettings[index][12];

calDatum.Add(caldata2);

// VNA calibration configuration 3

CalDataPortModule caldata3 = new CalDataPortModule();

caldata3.srcSelect = SigGen.SG1;

// Source Port
if (Enum.TryParse<SrcPort>(portSettings[index][1], true, out srcPort) == true)
{
caldata3.srcPort = srcPort;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][1], index + linePortDataStartsAt + 1, portSettings[index][1]));
}

// Source Port Alias
caldata3.srcPortAlias = portSettings[index][2];

if (caldata3.srcPort == SrcPort.P1) caldata3.measPort = MeasPort.P1_Fwd;
else if (caldata3.srcPort == SrcPort.P2) caldata3.measPort = MeasPort.P2_Fwd;
else if (caldata3.srcPort == SrcPort.P3) caldata3.measPort = MeasPort.P3_Fwd;
else if (caldata3.srcPort == SrcPort.P4) caldata3.measPort = MeasPort.P4_Fwd;
else if (caldata3.srcPort == SrcPort.P5) caldata3.measPort = MeasPort.P5_Fwd;
else if (caldata3.srcPort == SrcPort.P6) caldata3.measPort = MeasPort.P6_Fwd;
else if (caldata3.srcPort == SrcPort.P7) caldata3.measPort = MeasPort.P7_Fwd;
else if (caldata3.srcPort == SrcPort.P8) caldata3.measPort = MeasPort.P8_Fwd;
else throw new CalibrationDefinitionException(string.Format("Source Port Value Invalid : {0} - Line
{1}. (Valid values of P1 to P8.)", caldata3.srcPort, index + linePortDataStartsAt + 1));

// Measure Port Alias
caldata3.measPortAlias = portSettings[index][4];

// Frequency
caldata3.srcFreq = ConvertStringToDouble(portSettings[index][5], portSettings[0][5], index +
```

```
linePortDataStartsAt + 1);

// Level
caldata3.srcLevel = ConvertStringToDouble(portSettings[index][6], portSettings[0][6], index +
linePortDataStartsAt + 1);

// Filter
if (Enum.TryParse<MeasFilt>(portSettings[index][7], true, out measFilt) == true)
{
caldata3.measFilter = measFilt;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][7], index + linePortDataStartsAt + 1, portSettings[index][7]));
}

// Meas Attenuation
caldata3.measAtten = ConvertStringToDouble(portSettings[index][8], portSettings[0][8], index +
linePortDataStartsAt + 1);

// Modulation Type
caldata3.modulationType = portSettings[index][9];

// Modulation File
caldata3.modulationFile = portSettings[index][10];

// Waveform settings
caldata3.WaveConfig = new WaveformSettings();

// Marker
if (portSettings[index][11] != string.Empty)
{
if (portSettings[index][11].ToUpper() == "EXT")
{
caldata3.WaveConfig.Marker = 0;
}
else
{
caldata3.WaveConfig.Marker = ConvertStringToShort(portSettings[index][11], portSettings[0][11],
index + linePortDataStartsAt + 1);
}
```

```
}
else
{
caldata3.WaveConfig.Marker = -1;
}

// Analysis slot
if (portSettings[index][12] != string.Empty)
{
caldata3.WaveConfig.AnalysisSlot = ConvertStringToShort(portSettings[index][12],
portSettings[0][12], index + linePortDataStartsAt + 1);
}
else
{
caldata3.WaveConfig.AnalysisSlot = -1;
}

// Comment
caldata3.comment = portSettings[index][18];

calDatum.Add(caldata3);

// VNA calibration configuration 4

CalDataPortModule caldata4 = new CalDataPortModule();

caldata4.srcSelect = SigGen.SG1;

// Source Port
if (Enum.TryParse<SrcPort>(portSettings[index][1], true, out srcPort) == true)
{
caldata4.srcPort = srcPort;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][1], index + linePortDataStartsAt + 1, portSettings[index][1]));
}

// Source Port Alias
caldata4.srcPortAlias = portSettings[index][2];
```

```
if (caldata4.srcPort == SrcPort.P1) caldata4.measPort = MeasPort.P1_Rev;
else if (caldata4.srcPort == SrcPort.P2) caldata4.measPort = MeasPort.P2_Rev;
else if (caldata4.srcPort == SrcPort.P3) caldata4.measPort = MeasPort.P3_Rev;
else if (caldata4.srcPort == SrcPort.P4) caldata4.measPort = MeasPort.P4_Rev;
else if (caldata4.srcPort == SrcPort.P5) caldata4.measPort = MeasPort.P5_Rev;
else if (caldata4.srcPort == SrcPort.P6) caldata4.measPort = MeasPort.P6_Rev;
else if (caldata4.srcPort == SrcPort.P7) caldata4.measPort = MeasPort.P7_Rev;
else if (caldata4.srcPort == SrcPort.P8) caldata4.measPort = MeasPort.P8_Rev;
else throw new CalibrationDefinitionException(string.Format("Source Port Value Invalid : {0} - Line
{1}. (Valid values of P1 to P8.)", caldata4.srcPort, index + linePortDataStartsAt + 1));

// Measure Port Alias
caldata4.measPortAlias = portSettings[index][4];

// Frequency
caldata4.srcFreq = ConvertStringToDouble(portSettings[index][5], portSettings[0][5], index +
linePortDataStartsAt + 1);

// Level
caldata4.srcLevel = ConvertStringToDouble(portSettings[index][6], portSettings[0][6], index +
linePortDataStartsAt + 1);

// Filter
if (Enum.TryParse<MeasFilt>(portSettings[index][7], true, out measFilt) == true)
{
caldata4.measFilter = measFilt;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][7], index + linePortDataStartsAt + 1, portSettings[index][7]));
}

// Meas Attenuation
caldata4.measAtten = ConvertStringToDouble(portSettings[index][8], portSettings[0][8], index +
linePortDataStartsAt + 1);

// Modulation Type
caldata4.modulationType = portSettings[index][9];

// Modulation File
caldata4.modulationFile = portSettings[index][10];
```

```
// Waveform settings
caldata4.WaveConfig = new WaveformSettings();

// Marker
if (portSettings[index][11] != string.Empty)
{
if (portSettings[index][11].ToUpper() == "EXT")
{
caldata4.WaveConfig.Marker = 0;
}
else
{
caldata4.WaveConfig.Marker = ConvertStringToShort(portSettings[index][11], portSettings[0][11],
index + linePortDataStartsAt + 1);
}
}
else
{
caldata4.WaveConfig.Marker = -1;
}

// Analysis slot
if (portSettings[index][12] != string.Empty)
{
caldata4.WaveConfig.AnalysisSlot = ConvertStringToShort(portSettings[index][12],
portSettings[0][12], index + linePortDataStartsAt + 1);
}
else
{
caldata4.WaveConfig.AnalysisSlot = -1;
}


// Comment
caldata4.comment = portSettings[index][18];

calDatum.Add(caldata4);
}
}
else
{
```

```
// Source Port
if (Enum.TryParse<SrcPort>(portSettings[index][1], true, out srcPort) == true)
{
caldata.srcPort = srcPort;
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][1], index + linePortDataStartsAt + 1, portSettings[index][1]));
}

// Source Port Alias
caldata.srcPortAlias = portSettings[index][2];

// Measure Port
if (Enum.TryParse<MeasPort>(portSettings[index][3], true, out measPort) == true)
{
caldata.measPort = measPort;
}
else
{
caldata.measPort = MeasPort.NONE;
//throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][3], index + linePortDataStartsAt + 1, portSettings[index][3]));
}



// Measure Port Alias
caldata.measPortAlias = portSettings[index][4];

// Frequency
caldata.srcFreq = ConvertStringToDouble(portSettings[index][5], portSettings[0][5], index +
linePortDataStartsAt + 1);

// Level
caldata.srcLevel = ConvertStringToDouble(portSettings[index][6], portSettings[0][6], index +
linePortDataStartsAt + 1);

// Filter
if (Enum.TryParse<MeasFilt>(portSettings[index][7], true, out measFilt) == true)
{
caldata.measFilter = measFilt;
```

```
}
else
{
throw new CalibrationDefinitionException(string.Format("{0} - Line {1} Value of {2} is invalid. ",
portSettings[0][7], index + linePortDataStartsAt + 1, portSettings[index][7]));
}

// Meas Attenuation
caldata.measAtten = ConvertStringToDouble(portSettings[index][8], portSettings[0][8], index +
linePortDataStartsAt + 1);

// Modulation Type
caldata.modulationType = portSettings[index][9];

// Modulation File
caldata.modulationFile = portSettings[index][10];

// Waveform settings
caldata.WaveConfig = new WaveformSettings();

// Marker
if (portSettings[index][11] != string.Empty)
{
if (portSettings[index][11].ToUpper() == "EXT")
{
caldata.WaveConfig.Marker = 0;
}
else
{
caldata.WaveConfig.Marker = ConvertStringToShort(portSettings[index][11], portSettings[0][11],
index + linePortDataStartsAt + 1);
}
}
else
{
caldata.WaveConfig.Marker = -1;
}

// Analysis slot
if (portSettings[index][12] != string.Empty)
{
caldata.WaveConfig.AnalysisSlot = ConvertStringToShort(portSettings[index][12],
```

```
portSettings[0][12], index + linePortDataStartsAt + 1);
}
else
{
caldata.WaveConfig.AnalysisSlot = -1;
}

// Comment
caldata.comment = portSettings[index][18];

calDatum.Add(caldata);
}
}

if (string.IsNullOrWhiteSpace(info.Product) == true)
{
throw new CalibrationDefinitionException("Product Value not defined.");
}

if (string.IsNullOrWhiteSpace(info.CalVersion) == true)
{
throw new CalibrationDefinitionException("Calibration Version Value not defined.");
}

return Tuple.Create(info, CalSelect, attenExternal, calDatum);
}

//public Tuple<Info, CalSettings, ExternalAttenuation, List<CalDataPortModule>>
ImportCalConfigVer5(string calConfigFile)
//{
//    string line = "";
//    Info info = new Info();
//    CalSettings CalSelect = new CalSettings();
//    ExternalAttenuation attenExternal = new ExternalAttenuation();
//    List<CalDataPortModule> calDatum = new List<CalDataPortModule>();

//    // Initialize CalSelect
//    CalSelect.PowerMeterAvailable = true;
//    CalSelect.SourceMeasurePath = false;
//    CalSelect.InternalMatrixPath = true;
//    CalSelect.NoiseSource = false;
//    CalSelect.VNA = true;
```

```
//    CalSelect.NoiseBandwidth = 8e6;

//    try
//    {
//        using (StreamReader stream = new StreamReader(calConfigFile))
//        {
//            while ((line = stream.ReadLine()) != null)
//            {
//                string tmpString = line.Split(',')[0];

//                if (tmpString.ToUpper() == "PRODUCT:")
//                {
//                    info.Product = line.Split(',')[1];
//                }
//                if (tmpString.ToUpper() == "REVISION:")
//                {
//                    info.Revision = line.Split(',')[1];
//                }
//                if (tmpString.ToUpper() == "TEST PROGRAM:")
//                {
//                    info.Program = line.Split(',')[1];
//                }
//                if (tmpString.ToUpper() == "CALIBRATION VERSION:")
//                {
//                    info.CalVersion = line.Split(',')[1];
//                }
//                if (tmpString.ToUpper() == "COMMENT:")
//                {
//                    info.Comment = line.Split(',')[1];
//                }
//                else if (tmpString.ToUpper() == "CALIBRATE RF SOURCE/MEASURE:")
//                {
//                    CalSelect.SourceMeasurePath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
//                }
//                else if (tmpString.ToUpper() == "CALIBRATE NOISE SOURCE:")
//                {
//                    CalSelect.NoiseSource = line.Split(',')[1].ToUpper() == "Y" ? true : false;
//                }
//                else if (tmpString.ToUpper() == "CALIBRATE VNA:")
//                {
//                    CalSelect.VNA = line.Split(',')[1].ToUpper() == "Y" ? true : false;
//                }
```

```
//              else if (tmpString.ToUpper() == "NOISE BANDWIDTH (HZ):")
//              {
//                  CalSelect.NoiseBandwidth = double.Parse(line.Split(',')[1]);
//              }
//              else if (tmpString.ToUpper() == "MT-RF100 ATTENUATION (DB):")
//              {
//                  const int pathCount = 17;

//                  for (int srcIndex = 0; srcIndex < pathCount; srcIndex++)
//                  {
//                      attenExternal.srcAttenPortModule[srcIndex] = double.Parse(line.Split(',')[srcIndex
+ 1]);
//                  }
//              }
//              else if (tmpString.ToUpper() == "MT-RF200 ATTENUATION (DB):")
//              {
//                  const int pathCount = 30;

//                  for (int measIndex = 0; measIndex < pathCount; measIndex++)
//                  {
//                      attenExternal.measAttenPortModule[measIndex] =
double.Parse(line.Split(',')[measIndex + 1]);
//                  }
//              }
//              else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG2" ||
tmpString.ToUpper() == "NOISE")
//              {
//                  CalDataPortModule caldata = new CalDataPortModule();

//                  tmpString = line.Split(',')[0];
//                  if (tmpString.ToUpper() == "SG1") caldata.srcSelect = SigGen.SG1;
//                  else if (tmpString.ToUpper() == "SG2") caldata.srcSelect = SigGen.SG2;
//                  else if (tmpString.ToUpper() == "NOISE") caldata.srcSelect = SigGen.NOISE;

//                  tmpString = line.Split(',')[1];
//                  if (tmpString.ToUpper() == "P1") caldata.srcPort = SrcPort.P1;
//                  else if (tmpString.ToUpper() == "P2") caldata.srcPort = SrcPort.P2;
//                  else if (tmpString.ToUpper() == "P3") caldata.srcPort = SrcPort.P3;
//                  else if (tmpString.ToUpper() == "P4") caldata.srcPort = SrcPort.P4;
//                  else if (tmpString.ToUpper() == "P5") caldata.srcPort = SrcPort.P5;
//                  else if (tmpString.ToUpper() == "P6") caldata.srcPort = SrcPort.P6;
//                  else if (tmpString.ToUpper() == "P7") caldata.srcPort = SrcPort.P7;
```

```
//                  else if (tmpString.ToUpper() == "P8") caldata.srcPort = SrcPort.P8;
//                  else if (tmpString.ToUpper() == "P9") caldata.srcPort = SrcPort.P9;
//                  else if (tmpString.ToUpper() == "P10") caldata.srcPort = SrcPort.P10;
//                  else if (tmpString.ToUpper() == "P11") caldata.srcPort = SrcPort.P11;
//                  else if (tmpString.ToUpper() == "P12") caldata.srcPort = SrcPort.P12;
//                  else if (tmpString.ToUpper() == "P13") caldata.srcPort = SrcPort.P13;
//                  else if (tmpString.ToUpper() == "P14") caldata.srcPort = SrcPort.P14;
//                  else if (tmpString.ToUpper() == "P15") caldata.srcPort = SrcPort.P15;
//                  else if (tmpString.ToUpper() == "P16") caldata.srcPort = SrcPort.P16;
//                  else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG1_OUT")
caldata.srcPort = SrcPort.SG1_OUT;
//                  else if (tmpString.ToUpper() == "CAL") caldata.srcPort = SrcPort.CAL;
//                  else caldata.srcPort = SrcPort.NONE;

//                  caldata.srcPortAlias = line.Split(',')[2];

//                  tmpString = line.Split(',')[3];
//                  if (tmpString.ToUpper() == "P1") caldata.measPort = MeasPort.P1;
//                  else if (tmpString.ToUpper() == "P2") caldata.measPort = MeasPort.P2;
//                  else if (tmpString.ToUpper() == "P3") caldata.measPort = MeasPort.P3;
//                  else if (tmpString.ToUpper() == "P4") caldata.measPort = MeasPort.P4;
//                  else if (tmpString.ToUpper() == "P5") caldata.measPort = MeasPort.P5;
//                  else if (tmpString.ToUpper() == "P6") caldata.measPort = MeasPort.P6;
//                  else if (tmpString.ToUpper() == "P7") caldata.measPort = MeasPort.P7;
//                  else if (tmpString.ToUpper() == "P8") caldata.measPort = MeasPort.P8;
//                  else if (tmpString.ToUpper() == "P9") caldata.measPort = MeasPort.P9;
//                  else if (tmpString.ToUpper() == "P10") caldata.measPort = MeasPort.P10;
//                  else if (tmpString.ToUpper() == "P11") caldata.measPort = MeasPort.P11;
//                  else if (tmpString.ToUpper() == "P12") caldata.measPort = MeasPort.P12;
//                  else if (tmpString.ToUpper() == "P13") caldata.measPort = MeasPort.P13;
//                  else if (tmpString.ToUpper() == "P14") caldata.measPort = MeasPort.P14;
//                  else if (tmpString.ToUpper() == "P15") caldata.measPort = MeasPort.P15;
//                  else if (tmpString.ToUpper() == "P16") caldata.measPort = MeasPort.P16;
//                  else if (tmpString.ToUpper() == "P1_FWD") caldata.measPort = MeasPort.P1_Fwd;
//                  else if (tmpString.ToUpper() == "P2_FWD") caldata.measPort = MeasPort.P2_Fwd;
//                  else if (tmpString.ToUpper() == "P3_FWD") caldata.measPort = MeasPort.P3_Fwd;
//                  else if (tmpString.ToUpper() == "P4_FWD") caldata.measPort = MeasPort.P4_Fwd;
//                  else if (tmpString.ToUpper() == "P5_FWD") caldata.measPort = MeasPort.P5_Fwd;
//                  else if (tmpString.ToUpper() == "P6_FWD") caldata.measPort = MeasPort.P6_Fwd;
//                  else if (tmpString.ToUpper() == "P7_FWD") caldata.measPort = MeasPort.P7_Fwd;
//                  else if (tmpString.ToUpper() == "P8_FWD") caldata.measPort = MeasPort.P8_Fwd;
//                  else if (tmpString.ToUpper() == "P1_REV") caldata.measPort = MeasPort.P1_Rev;
```

```
//              else if (tmpString.ToUpper() == "P2_REV") caldata.measPort = MeasPort.P2_Rev;
//              else if (tmpString.ToUpper() == "P3_REV") caldata.measPort = MeasPort.P3_Rev;
//              else if (tmpString.ToUpper() == "P4_REV") caldata.measPort = MeasPort.P4_Rev;
//              else if (tmpString.ToUpper() == "P5_REV") caldata.measPort = MeasPort.P5_Rev;
//              else if (tmpString.ToUpper() == "P6_REV") caldata.measPort = MeasPort.P6_Rev;
//              else if (tmpString.ToUpper() == "P7_REV") caldata.measPort = MeasPort.P7_Rev;
//              else if (tmpString.ToUpper() == "P8_REV") caldata.measPort = MeasPort.P8_Rev;
//              else if (tmpString.ToUpper() == "M1") caldata.measPort = MeasPort.M1;
//              else if (tmpString.ToUpper() == "M2") caldata.measPort = MeasPort.M2;
//              else if (tmpString.ToUpper() == "M3") caldata.measPort = MeasPort.M3;
//              else if (tmpString.ToUpper() == "M4") caldata.measPort = MeasPort.M4;
//              else if (tmpString.ToUpper() == "M5") caldata.measPort = MeasPort.M5;
//              else if (tmpString.ToUpper() == "M6") caldata.measPort = MeasPort.M6;
//              else if (tmpString.ToUpper() == "M7") caldata.measPort = MeasPort.M7;
//              else if (tmpString.ToUpper() == "M8") caldata.measPort = MeasPort.M8;
//              else if (tmpString.ToUpper() == "M9") caldata.measPort = MeasPort.M9;
//              else if (tmpString.ToUpper() == "M10") caldata.measPort = MeasPort.M10;
//              else if (tmpString.ToUpper() == "M11") caldata.measPort = MeasPort.M11;
//              else if (tmpString.ToUpper() == "M12") caldata.measPort = MeasPort.M12;
//              else if (tmpString.ToUpper() == "M13") caldata.measPort = MeasPort.M13;
//              else if (tmpString.ToUpper() == "M14") caldata.measPort = MeasPort.M14;
//              else if (tmpString.ToUpper() == "M15") caldata.measPort = MeasPort.M15;
//              else if (tmpString.ToUpper() == "M16") caldata.measPort = MeasPort.M16;
//              else if (tmpString.ToUpper() == "M17") caldata.measPort = MeasPort.M17;
//              else if (tmpString.ToUpper() == "M18") caldata.measPort = MeasPort.M18;
//              else if (tmpString.ToUpper() == "M19") caldata.measPort = MeasPort.M19;
//              else if (tmpString.ToUpper() == "M20") caldata.measPort = MeasPort.M20;
//              else if (tmpString.ToUpper() == "M21") caldata.measPort = MeasPort.M21;
//              else if (tmpString.ToUpper() == "M22") caldata.measPort = MeasPort.M22;
//              else if (tmpString.ToUpper() == "M23") caldata.measPort = MeasPort.M23;
//              else if (tmpString.ToUpper() == "M24") caldata.measPort = MeasPort.M24;
//              else if (tmpString.ToUpper() == "M25") caldata.measPort = MeasPort.M25;
//              else if (tmpString.ToUpper() == "M26") caldata.measPort = MeasPort.M26;
//              else if (tmpString.ToUpper() == "MA") caldata.measPort = MeasPort.MA;
//              else if (tmpString.ToUpper() == "MB") caldata.measPort = MeasPort.MB;
//              else if (tmpString.ToUpper() == "MC") caldata.measPort = MeasPort.MC;
//              else if (tmpString.ToUpper() == "MD") caldata.measPort = MeasPort.MD;
//              else if (tmpString.ToUpper() == "CAL") caldata.measPort = MeasPort.CAL;
//              else if (tmpString.ToUpper() == "NONE") caldata.measPort = MeasPort.NONE;
//              else caldata.measPort = MeasPort.NONE;

//              caldata.measPortAlias = line.Split(',')[4];
```

```
//                caldata.srcFreq = double.Parse(line.Split(',')[5]);
//                caldata.srcLevel = double.Parse(line.Split(',')[6]);

//                tmpString = line.Split(',')[7];
//                if (tmpString.ToUpper() == "BYPASS") caldata.measFilter = MeasFilt.Bypass;
//                else if (tmpString.ToUpper() == "ATTEN") caldata.measFilter = MeasFilt.Atten;
//                else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1")
caldata.measFilter = MeasFilt.FILT1;
//                else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2")
caldata.measFilter = MeasFilt.FILT2;
//                else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3")
caldata.measFilter = MeasFilt.FILT3;
//                else caldata.measFilter = MeasFilt.Bypass; // Default

//                caldata.measAtten = double.Parse(line.Split(',')[8]);
//                caldata.modulationType = line.Split(',')[9];
//                caldata.modulationFile = line.Split(',')[10];

//                caldata.WaveConfig = new WaveformSettings();

//                tmpString = line.Split(',')[11]; // v5
//                if (tmpString != string.Empty)
//                {
//                    if (tmpString.ToUpper() == "EXT") caldata.WaveConfig.Marker = 0;
//                    else caldata.WaveConfig.Marker = Convert.ToInt16(tmpString);
//                }
//                else
//                {
//                    caldata.WaveConfig.Marker = -1;
//                }

//                tmpString = line.Split(',')[12]; // v5
//                if (tmpString != string.Empty)
//                {
//                    caldata.WaveConfig.AnalysisSlot = Convert.ToInt16(tmpString);
//                }
//                else
//                {
//                    caldata.WaveConfig.AnalysisSlot = -1;
//                }

//                caldata.comment = line.Split(',')[13];
```

```
//                calDatum.Add(caldata);
//            }
//            else if (tmpString.ToUpper() == "VNA-1P" || tmpString.ToUpper() == "VNA-2P")
//            {
//                // VNA calibration configuration 1

//                CalDataPortModule caldata1 = new CalDataPortModule();

//                caldata1.srcSelect = SigGen.SG1;

//                tmpString = line.Split(',')[1];
//                if (tmpString.ToUpper() == "P1") caldata1.srcPort = SrcPort.P1;
//                else if (tmpString.ToUpper() == "P2") caldata1.srcPort = SrcPort.P2;
//                else if (tmpString.ToUpper() == "P3") caldata1.srcPort = SrcPort.P3;
//                else if (tmpString.ToUpper() == "P4") caldata1.srcPort = SrcPort.P4;
//                else if (tmpString.ToUpper() == "P5") caldata1.srcPort = SrcPort.P5;
//                else if (tmpString.ToUpper() == "P6") caldata1.srcPort = SrcPort.P6;
//                else if (tmpString.ToUpper() == "P7") caldata1.srcPort = SrcPort.P7;
//                else if (tmpString.ToUpper() == "P8") caldata1.srcPort = SrcPort.P8;
//                else caldata1.srcPort = SrcPort.NONE;

//                if (caldata1.srcPort == SrcPort.P1) caldata1.measPort = MeasPort.P1_Fwd;
//                else if (caldata1.srcPort == SrcPort.P2) caldata1.measPort = MeasPort.P2_Fwd;
//                else if (caldata1.srcPort == SrcPort.P3) caldata1.measPort = MeasPort.P3_Fwd;
//                else if (caldata1.srcPort == SrcPort.P4) caldata1.measPort = MeasPort.P4_Fwd;
//                else if (caldata1.srcPort == SrcPort.P5) caldata1.measPort = MeasPort.P5_Fwd;
//                else if (caldata1.srcPort == SrcPort.P6) caldata1.measPort = MeasPort.P6_Fwd;
//                else if (caldata1.srcPort == SrcPort.P7) caldata1.measPort = MeasPort.P7_Fwd;
//                else if (caldata1.srcPort == SrcPort.P8) caldata1.measPort = MeasPort.P8_Fwd;

//                caldata1.srcPortAlias = line.Split(',')[2];
//                caldata1.measPortAlias = line.Split(',')[4];

//                caldata1.srcFreq = double.Parse(line.Split(',')[5]);
//                caldata1.srcLevel = double.Parse(line.Split(',')[6]);

//                tmpString = line.Split(',')[7];
//                if (tmpString.ToUpper() == "BYPASS") caldata1.measFilter = MeasFilt.Bypass;
//                else if (tmpString.ToUpper() == "ATTEN") caldata1.measFilter = MeasFilt.Atten;
//                else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1")
caldata1.measFilter = MeasFilt.FILT1;
```

```
//                  else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2")
caldata1.measFilter = MeasFilt.FILT2;
//                  else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3")
caldata1.measFilter = MeasFilt.FILT3;
//                  else caldata1.measFilter = MeasFilt.Bypass; // Default

//                  caldata1.measAtten = double.Parse(line.Split(',')[8]);
//                  caldata1.modulationType = line.Split(',')[9];
//                  caldata1.modulationFile = line.Split(',')[10];

//                  caldata1.WaveConfig = new WaveformSettings();

//                  tmpString = line.Split(',')[11]; // v5
//                  if (tmpString != string.Empty)
//                  {
//                      if (tmpString.ToUpper() == "EXT") caldata1.WaveConfig.Marker = 0;
//                      else caldata1.WaveConfig.Marker = Convert.ToInt16(tmpString);
//                  }
//                  else
//                  {
//                      caldata1.WaveConfig.Marker = -1;
//                  }

//                  tmpString = line.Split(',')[12]; // v5
//                  if (tmpString != string.Empty)
//                  {
//                      caldata1.WaveConfig.AnalysisSlot = Convert.ToInt16(tmpString);
//                  }
//                  else
//                  {
//                      caldata1.WaveConfig.AnalysisSlot = -1;
//                  }

//                  caldata1.comment = line.Split(',')[13];

//                  calDatum.Add(caldata1);

//                  tmpString = line.Split(',')[0];
//                  if (tmpString.ToUpper() == "VNA-2P")
//                  {
//                      // VNA calibration configuration 2
```

```
//                  CalDataPortModule caldata2 = new CalDataPortModule();

//                  caldata2.srcSelect = SigGen.SG1;

//                  tmpString = line.Split(',')[1];
//                  if (tmpString.ToUpper() == "P1") caldata2.srcPort = SrcPort.P1;
//                  else if (tmpString.ToUpper() == "P2") caldata2.srcPort = SrcPort.P2;
//                  else if (tmpString.ToUpper() == "P3") caldata2.srcPort = SrcPort.P3;
//                  else if (tmpString.ToUpper() == "P4") caldata2.srcPort = SrcPort.P4;
//                  else if (tmpString.ToUpper() == "P5") caldata2.srcPort = SrcPort.P5;
//                  else if (tmpString.ToUpper() == "P6") caldata2.srcPort = SrcPort.P6;
//                  else if (tmpString.ToUpper() == "P7") caldata2.srcPort = SrcPort.P7;
//                  else if (tmpString.ToUpper() == "P8") caldata2.srcPort = SrcPort.P8;
//                  else caldata2.srcPort = SrcPort.NONE;

//                  tmpString = line.Split(',')[3];
//                  if (tmpString.ToUpper() == "P1") caldata2.measPort = MeasPort.P1_Rev;
//                  else if (tmpString.ToUpper() == "P2") caldata2.measPort = MeasPort.P2_Rev;
//                  else if (tmpString.ToUpper() == "P3") caldata2.measPort = MeasPort.P3_Rev;
//                  else if (tmpString.ToUpper() == "P4") caldata2.measPort = MeasPort.P4_Rev;
//                  else if (tmpString.ToUpper() == "P5") caldata2.measPort = MeasPort.P5_Rev;
//                  else if (tmpString.ToUpper() == "P6") caldata2.measPort = MeasPort.P6_Rev;
//                  else if (tmpString.ToUpper() == "P7") caldata2.measPort = MeasPort.P7_Rev;
//                  else if (tmpString.ToUpper() == "P8") caldata2.measPort = MeasPort.P8_Rev;

//                  caldata2.srcPortAlias = line.Split(',')[2];
//                  caldata2.measPortAlias = line.Split(',')[4];

//                  caldata2.srcFreq = double.Parse(line.Split(',')[5]);
//                  caldata2.srcLevel = double.Parse(line.Split(',')[6]);

//                  tmpString = line.Split(',')[7];
//                  if (tmpString.ToUpper() == "BYPASS") caldata2.measFilter = MeasFilt.Bypass;
//                  else if (tmpString.ToUpper() == "ATTEN") caldata2.measFilter = MeasFilt.Atten;
//                  else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1")
caldata2.measFilter = MeasFilt.FILT1;
//                  else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2")
caldata2.measFilter = MeasFilt.FILT2;
//                  else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3")
caldata2.measFilter = MeasFilt.FILT3;
//                  else caldata2.measFilter = MeasFilt.Bypass; // Default
```

```
//                  caldata2.measAtten = double.Parse(line.Split(',')[8]);
//                  caldata2.modulationType = line.Split(',')[9];
//                  caldata2.modulationFile = line.Split(',')[10];

//                  caldata2.WaveConfig = new WaveformSettings();

//                  tmpString = line.Split(',')[11]; // v5
//                  if (tmpString != string.Empty)
//                  {
//                     if (tmpString.ToUpper() == "EXT") caldata2.WaveConfig.Marker = 0;
//                     else caldata2.WaveConfig.Marker = Convert.ToInt16(tmpString);
//                  }
//                  else
//                  {
//                     caldata2.WaveConfig.Marker = -1;
//                  }

//                  tmpString = line.Split(',')[12]; // v5
//                  if (tmpString != string.Empty)
//                  {
//                     caldata2.WaveConfig.AnalysisSlot = Convert.ToInt16(tmpString);
//                  }
//                  else
//                  {
//                     caldata2.WaveConfig.AnalysisSlot = -1;
//                  }

//                  caldata2.comment = line.Split(',')[13];

//                  calDatum.Add(caldata2);

//                  // VNA calibration configuration 3

//                  CalDataPortModule caldata3 = new CalDataPortModule();

//                  caldata3.srcSelect = SigGen.SG1;

//                  tmpString = line.Split(',')[3];
//                  if (tmpString.ToUpper() == "P1") caldata3.srcPort = SrcPort.P1;
//                  else if (tmpString.ToUpper() == "P2") caldata3.srcPort = SrcPort.P2;
//                  else if (tmpString.ToUpper() == "P3") caldata3.srcPort = SrcPort.P3;
//                  else if (tmpString.ToUpper() == "P4") caldata3.srcPort = SrcPort.P4;
```

```
//                  else if (tmpString.ToUpper() == "P5") caldata3.srcPort = SrcPort.P5;
//                  else if (tmpString.ToUpper() == "P6") caldata3.srcPort = SrcPort.P6;
//                  else if (tmpString.ToUpper() == "P7") caldata3.srcPort = SrcPort.P7;
//                  else if (tmpString.ToUpper() == "P8") caldata3.srcPort = SrcPort.P8;
//                  else caldata3.srcPort = SrcPort.NONE;

//                  if (caldata3.srcPort == SrcPort.P1) caldata3.measPort = MeasPort.P1_Fwd;
//                  else if (caldata3.srcPort == SrcPort.P2) caldata3.measPort = MeasPort.P2_Fwd;
//                  else if (caldata3.srcPort == SrcPort.P3) caldata3.measPort = MeasPort.P3_Fwd;
//                  else if (caldata3.srcPort == SrcPort.P4) caldata3.measPort = MeasPort.P4_Fwd;
//                  else if (caldata3.srcPort == SrcPort.P5) caldata3.measPort = MeasPort.P5_Fwd;
//                  else if (caldata3.srcPort == SrcPort.P6) caldata3.measPort = MeasPort.P6_Fwd;
//                  else if (caldata3.srcPort == SrcPort.P7) caldata3.measPort = MeasPort.P7_Fwd;
//                  else if (caldata3.srcPort == SrcPort.P8) caldata3.measPort = MeasPort.P8_Fwd;

//                  caldata3.srcPortAlias = line.Split(',')[2];
//                  caldata3.measPortAlias = line.Split(',')[4];

//                  caldata3.srcFreq = double.Parse(line.Split(',')[5]);
//                  caldata3.srcLevel = double.Parse(line.Split(',')[6]);

//                  tmpString = line.Split(',')[7];
//                  if (tmpString.ToUpper() == "BYPASS") caldata3.measFilter = MeasFilt.Bypass;
//                  else if (tmpString.ToUpper() == "ATTEN") caldata3.measFilter = MeasFilt.Atten;
//                  else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1")
caldata3.measFilter = MeasFilt.FILT1;
//                  else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2")
caldata3.measFilter = MeasFilt.FILT2;
//                  else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3")
caldata3.measFilter = MeasFilt.FILT3;
//                  else caldata3.measFilter = MeasFilt.Bypass; // Default

//                  caldata3.measAtten = double.Parse(line.Split(',')[8]);
//                  caldata3.modulationType = line.Split(',')[9];
//                  caldata3.modulationFile = line.Split(',')[10];

//                  caldata3.WaveConfig = new WaveformSettings();

//                  tmpString = line.Split(',')[11]; // v5
//                  if (tmpString != string.Empty)
//                  {
//                      if (tmpString.ToUpper() == "EXT") caldata3.WaveConfig.Marker = 0;
```

```
//                  else caldata3.WaveConfig.Marker = Convert.ToInt16(tmpString);
//              }
//              else
//              {
//                  caldata3.WaveConfig.Marker = -1;
//              }

//              tmpString = line.Split(',')[12]; // v5
//              if (tmpString != string.Empty)
//              {
//                  caldata3.WaveConfig.AnalysisSlot = Convert.ToInt16(tmpString);
//              }
//              else
//              {
//                  caldata3.WaveConfig.AnalysisSlot = -1;
//              }

//              caldata3.comment = line.Split(',')[13];

//              calDatum.Add(caldata3);

//              // VNA calibration configuration 4

//              CalDataPortModule caldata4 = new CalDataPortModule();

//              caldata4.srcSelect = SigGen.SG1;

//              tmpString = line.Split(',')[3];
//              if (tmpString.ToUpper() == "P1") caldata4.srcPort = SrcPort.P1;
//              else if (tmpString.ToUpper() == "P2") caldata4.srcPort = SrcPort.P2;
//              else if (tmpString.ToUpper() == "P3") caldata4.srcPort = SrcPort.P3;
//              else if (tmpString.ToUpper() == "P4") caldata4.srcPort = SrcPort.P4;
//              else if (tmpString.ToUpper() == "P5") caldata4.srcPort = SrcPort.P5;
//              else if (tmpString.ToUpper() == "P6") caldata4.srcPort = SrcPort.P6;
//              else if (tmpString.ToUpper() == "P7") caldata4.srcPort = SrcPort.P7;
//              else if (tmpString.ToUpper() == "P8") caldata4.srcPort = SrcPort.P8;
//              else caldata4.srcPort = SrcPort.NONE;

//              tmpString = line.Split(',')[1];
//              if (tmpString.ToUpper() == "P1") caldata4.measPort = MeasPort.P1_Rev;
//              else if (tmpString.ToUpper() == "P2") caldata4.measPort = MeasPort.P2_Rev;
//              else if (tmpString.ToUpper() == "P3") caldata4.measPort = MeasPort.P3_Rev;
```

```
//                  else if (tmpString.ToUpper() == "P4") caldata4.measPort = MeasPort.P4_Rev;
//                  else if (tmpString.ToUpper() == "P5") caldata4.measPort = MeasPort.P5_Rev;
//                  else if (tmpString.ToUpper() == "P6") caldata4.measPort = MeasPort.P6_Rev;
//                  else if (tmpString.ToUpper() == "P7") caldata4.measPort = MeasPort.P7_Rev;
//                  else if (tmpString.ToUpper() == "P8") caldata4.measPort = MeasPort.P8_Rev;

//                  caldata4.srcPortAlias = line.Split(',')[2];
//                  caldata4.measPortAlias = line.Split(',')[4];

//                  caldata4.srcFreq = double.Parse(line.Split(',')[5]);
//                  caldata4.srcLevel = double.Parse(line.Split(',')[6]);

//                  tmpString = line.Split(',')[7];
//                  if (tmpString.ToUpper() == "BYPASS") caldata4.measFilter = MeasFilt.Bypass;
//                  else if (tmpString.ToUpper() == "ATTEN") caldata4.measFilter = MeasFilt.Atten;
//                  else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1")
caldata4.measFilter = MeasFilt.FILT1;
//                  else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2")
caldata4.measFilter = MeasFilt.FILT2;
//                  else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3")
caldata4.measFilter = MeasFilt.FILT3;
//                  else caldata4.measFilter = MeasFilt.Bypass; // Default

//                  caldata4.measAtten = double.Parse(line.Split(',')[8]);
//                  caldata4.modulationType = line.Split(',')[9];
//                  caldata4.modulationFile = line.Split(',')[10];

//                  caldata4.WaveConfig = new WaveformSettings();

//                  tmpString = line.Split(',')[11]; // v5
//                  if (tmpString != string.Empty)
//                  {
//                      if (tmpString.ToUpper() == "EXT") caldata4.WaveConfig.Marker = 0;
//                      else caldata4.WaveConfig.Marker = Convert.ToInt16(tmpString);
//                  }
//                  else
//                  {
//                      caldata4.WaveConfig.Marker = -1;
//                  }

//                  tmpString = line.Split(',')[12]; // v5
//                  if (tmpString != string.Empty)
```

```
//                  {
//                      caldata4.WaveConfig.AnalysisSlot = Convert.ToInt16(tmpString);
//                  }
//                  else
//                  {
//                      caldata4.WaveConfig.AnalysisSlot = -1;
//                  }

//                  caldata4.comment = line.Split(',')[13];

//                  calDatum.Add(caldata4);
//              }
//          }
//      }
//  }
//  catch (FileNotFoundException error)
//  {
//      Console.WriteLine("\n{0}", error.Message);
//  }

//  // Added for dynamic waveform calibration support (User Calibration v5.0 and up)
//  ReadWaveSettingsFile(ref calDatum);

//  return Tuple.Create(info, CalSelect, attenExternal, calDatum);
//}



private int CheckWaveSettingsFile(string file)
{
if (!File.Exists(file))
{
string message = "ERROR: Modulation waveform settings file not found:" + file + "\n";
Console.WriteLine(message);
//MessageBox.Show(message, "Merlin Test Technologies", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);
return -1;
}

// Check if file is locked
bool fileLocked = true;
```

```
do
{
try
{
FileStream fs = File.Open(file, FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.None);
fs.Close();
fileLocked = false;
}
catch (IOException ex)
{
MessageBox.Show(ex.Message + "\n\nPlease close modulation waveform settings file to
continue...", "Merlin Test Technologies", MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1);
}
} while (fileLocked == true);

return 0;
}

//public int ReadWaveSettingsFile(ref List<CalDataPortModule> calDatum)
//{
//    int status = -1;
//    string waveDirectory = @"C:\MerlinTest\Waveform Library\";

//    //-------------------- Read Modulation Waveform Paramaters From Waveform Settings File ---------
-----------

//    for (int calIndex = 0; calIndex < calDatum.Count; calIndex++)
//    {
//        if (calDatum[calIndex].modulationFile == "" || calDatum[calIndex].modulationType ==
"NONE" || calDatum[calIndex].modulationType == "CW")
//        {
//            if (calDatum[calIndex].modulationType == "" || calDatum[calIndex].modulationType ==
"NONE")
//            {
//                calDatum[calIndex].WaveConfig.WaveformFile = "";
//                calDatum[calIndex].WaveConfig.Modulation = ModulationType.None;
//            }
//            else if (calDatum[calIndex].modulationType == "CW")
//            {
//                calDatum[calIndex].WaveConfig.WaveformFile = calDatum[calIndex].modulationFile;
//                calDatum[calIndex].WaveConfig.Modulation = ModulationType.CW;
```

```
//          }
//          calDatum[calIndex].WaveConfig.Bandwidth = ChannelBandwidth.NA;
//          calDatum[calIndex].WaveConfig.Marker = -1;
//          calDatum[calIndex].WaveConfig.MarkerLocation = 0;
//          calDatum[calIndex].WaveConfig.DataLocation = 0;
//          calDatum[calIndex].WaveConfig.DataLength = 0.5e-3;
//          calDatum[calIndex].WaveConfig.MeasOffset =
calDatum[calIndex].WaveConfig.DataLocation - calDatum[calIndex].WaveConfig.MarkerLocation;
//          calDatum[calIndex].WaveConfig.MeasLength =
calDatum[calIndex].WaveConfig.MeasOffset + calDatum[calIndex].WaveConfig.DataLength;
//          calDatum[calIndex].WaveConfig.FrameLength =
calDatum[calIndex].WaveConfig.MeasOffset + calDatum[calIndex].WaveConfig.DataLength;
//          calDatum[calIndex].WaveConfig.DutyCycle = 100;
//          calDatum[calIndex].WaveConfig.MinSampleFreq = 1e6;
//          calDatum[calIndex].WaveConfig.RecSampleFreq = 1e6;
//          calDatum[calIndex].WaveConfig.MeasSpan = 0;
//          calDatum[calIndex].WaveConfig.Headroom = 0;
//      }
//      else if (calDatum[calIndex].modulationType == "GSM" || calDatum[calIndex].modulationType
== "EDGE")
//      {
//          try
//          {
//              string aiqFile = calDatum[calIndex].modulationFile;
//              string iqsFile = waveDirectory + "Waveforms\\" +
Path.GetFileNameWithoutExtension(aiqFile) + ".iqs";

//              // Check if .iqs file exists for corresponding .aiq file
//              status = CheckWaveSettingsFile(iqsFile);

//              //////// Need to put an error here if file doesn'e exist ////////

//              // Get waveform data
//              Waveform GsmWaveform = new Waveform();
//              GsmWaveformInfo GsmInfo = GsmWaveform.GetGsmEdgeWaveformInfo(iqsFile);

//              const int maxSlots = 8;
//              double symbolRate = GsmInfo.SymbolRate; //270.8333333e3;
//              double T = 3 / symbolRate; // Number of bits (T) / Symbol Rate
//              double slotLengthLong = (3 + 58 + 26 + 58 + 3 + 9) / symbolRate;  // Number of bits (T
+ Data + TSC + Data + T + G) / Symbol Rate
//              double slotLengthShort = (3 + 58 + 26 + 58 + 3 + 8) / symbolRate; // Number of bits (T
```

```
+ Data + TSC + Data + T + G) / Symbol Rate
//              double frameLength = (2 * slotLengthLong) + (6 * slotLengthShort);

//              double[] slotLocation = new double[maxSlots];
//              slotLocation[0] = 0;
//              slotLocation[1] = slotLengthLong;
//              slotLocation[2] = slotLengthLong + slotLengthShort;
//              slotLocation[3] = slotLengthLong + (2 * slotLengthShort);
//              slotLocation[4] = slotLengthLong + (3 * slotLengthShort);
//              slotLocation[5] = (2 * slotLengthLong) + (3 * slotLengthShort);
//              slotLocation[6] = (2 * slotLengthLong) + (4 * slotLengthShort);
//              slotLocation[7] = (2 * slotLengthLong) + (5 * slotLengthShort);

//              double[] dataLocation = new double[maxSlots];
//              dataLocation[0] = slotLocation[0] + T;
//              dataLocation[1] = slotLocation[1] + T;
//              dataLocation[2] = slotLocation[2] + T;
//              dataLocation[3] = slotLocation[3] + T;
//              dataLocation[4] = slotLocation[4] + T;
//              dataLocation[5] = slotLocation[5] + T;
//              dataLocation[6] = slotLocation[6] + T;
//              dataLocation[7] = slotLocation[7] + T;

//              // Check whether specified waveform marker is valid
//              if (calDatum[calIndex].WaveConfig.Marker >= 1 &&
calDatum[calIndex].WaveConfig.Marker <= 4) // Marker 1 - 4
//              {
//                  if ((calDatum[calIndex].WaveConfig.Marker == 1 &&
GsmInfo.Marker.Marker1Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 2 &&
GsmInfo.Marker.Marker2Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 3 &&
GsmInfo.Marker.Marker3Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 4 &&
GsmInfo.Marker.Marker4Info.Count == 0))
//                  {
//                      Console.WriteLine("ERROR: Specified waveform marker {0} is undefined for
waveform {1}.", calDatum[calIndex].WaveConfig.Marker, aiqFile);
//                  }
//              }
//              else if (calDatum[calIndex].WaveConfig.Marker == -1) // No marker specified; default to
1st available marker
//              {
//                  if (GsmInfo.Marker.Marker1Info.Count != 0) calDatum[calIndex].WaveConfig.Marker
= 1;
```

```
//                  else if (GsmInfo.Marker.Marker2Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 2;
//                  else if (GsmInfo.Marker.Marker3Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 3;
//                  else if (GsmInfo.Marker.Marker4Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 4;
//                  else // Error
//                  {
//                      Console.WriteLine("ERROR: No waveform marker(s) defined for waveform {0}.",
aiqFile);
//                  }
//              }
//              else if (calDatum[calIndex].WaveConfig.Marker != 0) // Marker 0 = external marker
//              {
//                  Console.WriteLine("ERROR: Invalid waveform marker {0} for waveform {1}.",
calDatum[calIndex].WaveConfig.Marker, aiqFile);
//              }

//              // Determine number of triggers for specified marker
//              int numOfTriggers = 0;
//              if (calDatum[calIndex].WaveConfig.Marker == 0) numOfTriggers = 1;
//              else if (calDatum[calIndex].WaveConfig.Marker == 1) numOfTriggers =
GsmInfo.Marker.Marker1Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 2) numOfTriggers =
GsmInfo.Marker.Marker2Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 3) numOfTriggers =
GsmInfo.Marker.Marker3Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 4) numOfTriggers =
GsmInfo.Marker.Marker4Info.Count;

//              // Check whether to use specific slot for capture/analysis
//              bool analyzeSpecificSlot = false;
//              if (calDatum[calIndex].WaveConfig.AnalysisSlot != -1)
//              {
//                  if (calDatum[calIndex].WaveConfig.AnalysisSlot >= 1 &&
calDatum[calIndex].WaveConfig.AnalysisSlot <= maxSlots)
//                  {
//                      analyzeSpecificSlot = true;
//                  }
//                  else // Error
//                  {
//                      Console.WriteLine("Requested analysis slot {0} out of range for GSM/EDGE (valid
```

```
slot is 1 - {1}).", calDatum[calIndex].WaveConfig.AnalysisSlot, maxSlots);
//                 }
//             }

//             // Determine how many slots are enabled
//             int numOfSlotsEnabled = 0;
//             bool specificSlotEnabled = false;
//             for (int slot = 0; slot < maxSlots; slot++)
//             {
//                 if (GsmInfo.SlotEnabled[slot] == true)
//                 {
//                     numOfSlotsEnabled++;
//                     if (slot + 1 == calDatum[calIndex].WaveConfig.AnalysisSlot) specificSlotEnabled =
true;
//                 }
//             }

//             if (numOfSlotsEnabled < 1) // Error
//             {
//                 Console.WriteLine("ERROR: No slots enabled for waveform {0}.", aiqFile);
//             }

//             // Generate list of enabled slots
//             int index = 0;
//             int[] enabledSlots = new int[numOfSlotsEnabled];
//             for (int slotIndex = 0; slotIndex < maxSlots; slotIndex++)
//             {
//                 if (GsmInfo.SlotEnabled[slotIndex] == true)
//                 {
//                     enabledSlots[index++] = slotIndex + 1;
//                 }
//             }

//             if (analyzeSpecificSlot && !specificSlotEnabled) // Error
//             {
//                 Console.WriteLine("ERROR: Specified analysis slot {0} is not enabled for waveform
{1}.", calDatum[calIndex].WaveConfig.AnalysisSlot, aiqFile);
//             }

//             double triggerLocation = 0;
//             double[] offsets = new double[numOfTriggers];
```

```
//              if (numOfTriggers == 0)
//              {
//                  Console.WriteLine("ERROR: No valid trigger available for waveform {0}.", aiqFile);
//              }
//              if (numOfTriggers == 1 && calDatum[calIndex].WaveConfig.Marker == 0)
//              {
//                  calDatum[calIndex].WaveConfig.MeasOffset = 0; // Assumes extermal trigger occurs
at enabled slot location
//              }
//              else if (numOfTriggers == 1)
//              {
//                  if (calDatum[calIndex].WaveConfig.Marker == 1) triggerLocation =
GsmInfo.Marker.Marker1Info[0].OnTime / symbolRate;
//                  else if (calDatum[calIndex].WaveConfig.Marker == 2) triggerLocation =
GsmInfo.Marker.Marker2Info[0].OnTime / symbolRate;
//                  else if (calDatum[calIndex].WaveConfig.Marker == 3) triggerLocation =
GsmInfo.Marker.Marker3Info[0].OnTime / symbolRate;
//                  else if (calDatum[calIndex].WaveConfig.Marker == 4) triggerLocation =
GsmInfo.Marker.Marker4Info[0].OnTime / symbolRate;

//                  if (analyzeSpecificSlot)
//                  {
//                      offsets[0] = dataLocation[calDatum[calIndex].WaveConfig.AnalysisSlot - 1] -
triggerLocation;
//                      if (offsets[0] < 0) offsets[0] += frameLength; // TODO: Need to check if negative
offsets are valid
//                  }
//                  else
//                  {
//                      bool slotFound = false;

//                      // Find offset from trigger location to first enabled slot
//                      for (int slotIndex = 0; slotIndex < enabledSlots.Length; slotIndex++)
//                      {
//                          if (slotLocation[enabledSlots[slotIndex] - 1] >= triggerLocation ||
dataLocation[enabledSlots[slotIndex] - 1] >= triggerLocation)
//                          {
//                              offsets[0] = dataLocation[enabledSlots[slotIndex] - 1] - triggerLocation;
//                              slotFound = true;
//                          }

//                          if (slotFound) break;
```

```
//                  }

//                  // Find offset from trigger location to first enabled slot preceeding the trigger
(uncommon)
//                  if (!slotFound)
//                  {
//                      offsets[0] = (dataLocation[enabledSlots[0] - 1] - triggerLocation) + frameLength;
//                      slotFound = true;
//                  }
//              }

//              calDatum[calIndex].WaveConfig.MeasOffset = offsets[0];
//          }
//          else if (numOfTriggers == numOfSlotsEnabled) // Assumes each trigger marks the
location of an enabled slot
//          {
//              // Calculate trigger offsets
//              for (int slotIndex = 0; slotIndex < enabledSlots.Length; slotIndex++)
//              {
//                  if (calDatum[calIndex].WaveConfig.Marker == 1) offsets[slotIndex] =
dataLocation[enabledSlots[slotIndex] - 1] - (GsmInfo.Marker.Marker1Info[slotIndex].OnTime /
symbolRate);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 2) offsets[slotIndex] =
dataLocation[enabledSlots[slotIndex] - 1] - (GsmInfo.Marker.Marker2Info[slotIndex].OnTime /
symbolRate);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 3) offsets[slotIndex] =
dataLocation[enabledSlots[slotIndex] - 1] - (GsmInfo.Marker.Marker3Info[slotIndex].OnTime /
symbolRate);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 4) offsets[slotIndex] =
dataLocation[enabledSlots[slotIndex] - 1] - (GsmInfo.Marker.Marker4Info[slotIndex].OnTime /
symbolRate);
//              }

//              // Remove unnecessary digits from small double-precision value
//              double[] tmpOffsets = new double[offsets.Length];
//              for (int i = 0; i < offsets.Length; i++)
//              {
//                  tmpOffsets[i] = Math.Round(offsets[i], 12); // Round to picoseconds
//              }

//              // Check if trigger offsets are consistent
//              bool offsetMismatch = false;
```

```
//                 if (tmpOffsets.Length > 1)
//                 {
//                     for (int i = 1; i < tmpOffsets.Length; i++)
//                     {
//                         if (tmpOffsets[i] != tmpOffsets[i - 1]) offsetMismatch = true;
//                     }
//                 }

//                 // All trigger offsets are uniform, so set MeasOffset to any offset value
//                 if (!offsetMismatch)
//                 {
//                     calDatum[calIndex].WaveConfig.MeasOffset = tmpOffsets[0];
//                 }
//                 else
//                 {
//                     // Error
//                 }
//             }
//             else if (numOfTriggers < numOfSlotsEnabled)
//             {
//                 double[,] offsetsAll = new double[numOfTriggers, numOfSlotsEnabled];

//                 // Calculate all offsets from trigger location to each enabled slot
//                 for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//                 {
//                     for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                     {
//                         if (calDatum[calIndex].WaveConfig.Marker == 1) offsetsAll[trigIndex, slotIndex]
= dataLocation[enabledSlots[slotIndex] - 1] - (GsmInfo.Marker.Marker1Info[trigIndex].OnTime /
symbolRate);
//                         else if (calDatum[calIndex].WaveConfig.Marker == 2) offsetsAll[trigIndex,
slotIndex] = dataLocation[enabledSlots[slotIndex] - 1] -
(GsmInfo.Marker.Marker2Info[trigIndex].OnTime / symbolRate);
//                         else if (calDatum[calIndex].WaveConfig.Marker == 3) offsetsAll[trigIndex,
slotIndex] = dataLocation[enabledSlots[slotIndex] - 1] -
(GsmInfo.Marker.Marker3Info[trigIndex].OnTime / symbolRate);
//                         else if (calDatum[calIndex].WaveConfig.Marker == 4) offsetsAll[trigIndex,
slotIndex] = dataLocation[enabledSlots[slotIndex] - 1] -
(GsmInfo.Marker.Marker4Info[trigIndex].OnTime / symbolRate);

//                         offsetsAll[trigIndex, slotIndex] = Math.Round(offsetsAll[trigIndex, slotIndex], 9);
// Round to picoseconds
```

```
//                  }
//              }

//              // Find first valid (non-negative) offset for each trigger pulse
//              int[] offsetPosition = new int[numOfTriggers];
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (offsetsAll[trigIndex, slotIndex] >= 0)
//                      {
//                          offsetPosition[trigIndex] = slotIndex;
//                          break;
//                      }
//                  }
//              }

//              // Arrange/align offsets starting with valid (non-negative) values
//              double[,] tmpOffsetsAllAligned = new double[numOfTriggers, numOfSlotsEnabled];
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  int slotPosition = offsetPosition[trigIndex];

//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (slotPosition >= numOfSlotsEnabled) slotPosition = 0;
//                      tmpOffsetsAllAligned[trigIndex, slotIndex] = offsetsAll[trigIndex, slotPosition];
//                      slotPosition++;
//                  }
//              }

//              // Convert negative offsets to positive values
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (tmpOffsetsAllAligned[trigIndex, slotIndex] < 0)
tmpOffsetsAllAligned[trigIndex, slotIndex] += frameLength;
//                  }
//              }

//              // Remove unnecessary digits from small double-precision value
```

```
//                for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//                {
//                    for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                    {
//                        tmpOffsetsAllAligned[trigIndex, slotIndex] =
Math.Round(tmpOffsetsAllAligned[trigIndex, slotIndex], 9); // Round to picoseconds
//                    }
//                }

//                // Store trigger offsets for enabled slots within trigger(n) and trigger(n+1)
//                double trigPeriod = 0;
//                double trigPeriodMax = frameLength;
//                for (int trigIndex = 1; trigIndex <= numOfTriggers; trigIndex++)
//                {
//                    if (trigIndex < numOfTriggers) // Calculate amount of time between adjacent
triggers
//                    {
//                        if (calDatum[calIndex].WaveConfig.Marker == 1) trigPeriod =
(GsmInfo.Marker.Marker1Info[trigIndex].OnTime - GsmInfo.Marker.Marker1Info[trigIndex -
1].OnTime) / symbolRate;
//                        else if (calDatum[calIndex].WaveConfig.Marker == 2) trigPeriod =
(GsmInfo.Marker.Marker2Info[trigIndex].OnTime - GsmInfo.Marker.Marker2Info[trigIndex -
1].OnTime) / symbolRate;
//                        else if (calDatum[calIndex].WaveConfig.Marker == 3) trigPeriod =
(GsmInfo.Marker.Marker3Info[trigIndex].OnTime - GsmInfo.Marker.Marker3Info[trigIndex -
1].OnTime) / symbolRate;
//                        else if (calDatum[calIndex].WaveConfig.Marker == 4) trigPeriod =
(GsmInfo.Marker.Marker4Info[trigIndex].OnTime - GsmInfo.Marker.Marker4Info[trigIndex -
1].OnTime) / symbolRate;
//                    }
//                    else if (trigIndex == numOfTriggers) // Calculate amount of time between last
trigger and first trigger (waveform wrap-around)
//                    {
//                        if (calDatum[calIndex].WaveConfig.Marker == 1) trigPeriod = frameLength -
(GsmInfo.Marker.Marker1Info[trigIndex - 1].OnTime / symbolRate) +
(GsmInfo.Marker.Marker1Info[0].OnTime / symbolRate);
//                        else if (calDatum[calIndex].WaveConfig.Marker == 2) trigPeriod = frameLength
- (GsmInfo.Marker.Marker2Info[trigIndex - 1].OnTime / symbolRate) +
(GsmInfo.Marker.Marker2Info[0].OnTime / symbolRate);
//                        else if (calDatum[calIndex].WaveConfig.Marker == 3) trigPeriod = frameLength
- (GsmInfo.Marker.Marker3Info[trigIndex - 1].OnTime / symbolRate) +
(GsmInfo.Marker.Marker3Info[0].OnTime / symbolRate);
```

```
//                      else if (calDatum[calIndex].WaveConfig.Marker == 4) trigPeriod = frameLength
- (GsmInfo.Marker.Marker4Info[trigIndex - 1].OnTime / symbolRate) +
(GsmInfo.Marker.Marker4Info[0].OnTime / symbolRate);
//                  }

//                  trigPeriod = Math.Round(trigPeriod, 9); // Round to picoseconds
//                  if (trigPeriod < trigPeriodMax) trigPeriodMax = trigPeriod;
//              }

//              // Ignore offsets that are beyond the next consecutive trigger
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (tmpOffsetsAllAligned[trigIndex, slotIndex] >= trigPeriodMax)
tmpOffsetsAllAligned[trigIndex, slotIndex] = -9999;
//                  }
//              }

//              // Check if trigger offsets are consistent
//              bool offsetMismatch = false;
//              if (tmpOffsetsAllAligned.GetLength(0) > 1)
//              {
//                  for (int trigIndex = 1; trigIndex < tmpOffsetsAllAligned.GetLength(0); trigIndex++)
//                  {
//                      for (int slotIndex = 0; slotIndex < tmpOffsetsAllAligned.GetLength(1);
slotIndex++)
//                      {
//                          if (tmpOffsetsAllAligned[trigIndex, slotIndex] != -9999 &&
tmpOffsetsAllAligned[trigIndex, slotIndex] != tmpOffsetsAllAligned[trigIndex - 1, slotIndex])
offsetMismatch = true;
//                      }
//                  }
//              }

//              // All trigger offsets are the same, so set MeasOffset to first enabled slot
//              if (!offsetMismatch)
//              {
//                  calDatum[calIndex].WaveConfig.MeasOffset = tmpOffsetsAllAligned[0, 0];
//              }
//              else
//              {
```

```
//                      // Error
//                  }
//              }

//              if (calDatum[calIndex].modulationType == "GSM")
calDatum[calIndex].WaveConfig.Modulation = ModulationType.GSM;
//              else if (calDatum[calIndex].modulationType == "EDGE")
calDatum[calIndex].WaveConfig.Modulation = ModulationType.EDGE;
//              calDatum[calIndex].WaveConfig.WaveformFile = aiqFile;
//              calDatum[calIndex].WaveConfig.Bandwidth = ChannelBandwidth.NA; // 200 kHz
//              calDatum[calIndex].WaveConfig.FrameLength = frameLength; //4.615e-3;
//              calDatum[calIndex].WaveConfig.DataLength = (58 + 26 + 58) / symbolRate; // Number
of bits (Data + TSC + Data) / Symbol Rate
//              calDatum[calIndex].WaveConfig.MeasLength =
calDatum[calIndex].WaveConfig.MeasOffset + calDatum[calIndex].WaveConfig.DataLength;
//              calDatum[calIndex].WaveConfig.DutyCycle =
(calDatum[calIndex].WaveConfig.DataLength * numOfSlotsEnabled / frameLength) * 100;
//              calDatum[calIndex].WaveConfig.MinSampleFreq = 4.333333e6;
//              calDatum[calIndex].WaveConfig.RecSampleFreq = 4.333333e6;
//              calDatum[calIndex].WaveConfig.MeasSpan = 0;
//              calDatum[calIndex].WaveConfig.Headroom = 3;
//          }
//          catch (FileNotFoundException error)
//          {
//              Console.WriteLine("\n{0}", error.Message);
//              return -1;
//          }
//      }
//      else if (calDatum[calIndex].modulationType == "LTE-FDD" ||
calDatum[calIndex].modulationType == "LTE-TDD")
//      {
//          try
//          {
//              string aiqFile = calDatum[calIndex].modulationFile;
//              string iqsFile = waveDirectory + "Waveforms\\" +
Path.GetFileNameWithoutExtension(aiqFile) + ".iqs";

//              // Check if .iqs file exists for corresponding .aiq file
//              status = CheckWaveSettingsFile(iqsFile);

//              //////// Need to put an error here if file doesn'e exist ////////
```

```
//              if (calDatum[calIndex].modulationType == "LTE-FDD")
calDatum[calIndex].WaveConfig.Modulation = ModulationType.LTE_FDD;
//              else if (calDatum[calIndex].modulationType == "LTE-TDD")
calDatum[calIndex].WaveConfig.Modulation = ModulationType.LTE_TDD;

//              // Get waveform data
//              Waveform LteWaveform = new Waveform();
//              LteWaveformInfo LteInfo = LteWaveform.GetLteWaveformInfo(iqsFile,
calDatum[calIndex].WaveConfig.Modulation);

//              const uint Ts = 307200;
//              const ushort maxSlots = 20;
//              const ushort maxSubframes = 10;
//              const double slotLength = 0.5e-3;
//              const double subFrameLength = 1e-3;
//              const double frameLength = 10e-3;

//              double[] slotLocation = new double[maxSlots];
//              slotLocation[0] = 0;
//              slotLocation[1] = 0.0005;
//              slotLocation[2] = 0.001;
//              slotLocation[3] = 0.0015;
//              slotLocation[4] = 0.002;
//              slotLocation[5] = 0.0025;
//              slotLocation[6] = 0.003;
//              slotLocation[7] = 0.0035;
//              slotLocation[8] = 0.004;
//              slotLocation[9] = 0.0045;
//              slotLocation[10] = 0.005;
//              slotLocation[11] = 0.0055;
//              slotLocation[12] = 0.006;
//              slotLocation[13] = 0.0065;
//              slotLocation[14] = 0.007;
//              slotLocation[15] = 0.0075;
//              slotLocation[16] = 0.008;
//              slotLocation[17] = 0.0085;
//              slotLocation[18] = 0.009;
//              slotLocation[19] = 0.0095;

//              // Check whether specified waveform marker is valid
//              if (calDatum[calIndex].WaveConfig.Marker >= 1 &&
calDatum[calIndex].WaveConfig.Marker <= 4) // Marker 1 - 4
```

```
//              {
//                  if ((calDatum[calIndex].WaveConfig.Marker == 1 &&
LteInfo.Marker.Marker1Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 2 &&
LteInfo.Marker.Marker2Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 3 &&
LteInfo.Marker.Marker3Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 4 &&
LteInfo.Marker.Marker4Info.Count == 0))
//                  {
//                      Console.WriteLine("ERROR: Specified waveform marker {0} is undefined for
waveform {1}.", calDatum[calIndex].WaveConfig.Marker, aiqFile);
//                  }
//              }
//              else if (calDatum[calIndex].WaveConfig.Marker == -1) // No marker specified; default to
1st available marker
//              {
//                  if (LteInfo.Marker.Marker1Info.Count != 0) calDatum[calIndex].WaveConfig.Marker =
1;
//                  else if (LteInfo.Marker.Marker2Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 2;
//                  else if (LteInfo.Marker.Marker3Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 3;
//                  else if (LteInfo.Marker.Marker4Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 4;
//                  else // Error
//                  {
//                      Console.WriteLine("ERROR: No waveform marker(s) defined for waveform {0}.",
aiqFile);
//                  }
//              }
//              else if (calDatum[calIndex].WaveConfig.Marker != 0) // Marker 0 = external marker
//              {
//                  Console.WriteLine("ERROR: Invalid waveform marker {0} for waveform {1}.",
calDatum[calIndex].WaveConfig.Marker, aiqFile);
//              }

//              // Determine number of triggers for specified marker
//              int numOfTriggers = 0;
//              if (calDatum[calIndex].WaveConfig.Marker == 0) numOfTriggers = 1;
//              else if (calDatum[calIndex].WaveConfig.Marker == 1) numOfTriggers =
LteInfo.Marker.Marker1Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 2) numOfTriggers =
LteInfo.Marker.Marker2Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 3) numOfTriggers =
```

```
LteInfo.Marker.Marker3Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 4) numOfTriggers =
LteInfo.Marker.Marker4Info.Count;

//              // Check whether to use specific slot for capture/analysis
//              bool analyzeSpecificSlot = false;
//              if (calDatum[calIndex].WaveConfig.AnalysisSlot != -1)
//              {
//                  if (calDatum[calIndex].WaveConfig.AnalysisSlot >= 1 &&
calDatum[calIndex].WaveConfig.AnalysisSlot <= maxSlots)
//                  {
//                      analyzeSpecificSlot = true;
//                  }
//                  else // Error
//                  {
//                      Console.WriteLine("Requested analysis slot {0} out of range for LTE (valid slot is
1 - {1}).", calDatum[calIndex].WaveConfig.AnalysisSlot, maxSlots);
//                  }
//              }

//              // Determine how many slots are enabled
//              int numOfSlotsEnabled = 0;
//              bool specificSlotEnabled = false;
//              for (int slot = 0; slot < maxSlots; slot++)
//              {
//                  if (LteInfo.SlotEnabled[slot] == true)
//                  {
//                      numOfSlotsEnabled++;
//                      if (slot + 1 == calDatum[calIndex].WaveConfig.AnalysisSlot) specificSlotEnabled =
true;
//                  }
//              }

//              if (numOfSlotsEnabled < 1) // Error
//              {
//                  Console.WriteLine("ERROR: No slots enabled for waveform {0}.", aiqFile);
//              }

//              // Generate list of enabled slots
//              int index = 0;
//              int[] enabledSlots = new int[numOfSlotsEnabled];
//              for (int slotIndex = 0; slotIndex < maxSlots; slotIndex++)
```

```
//              {
//                  if (LteInfo.SlotEnabled[slotIndex] == true)
//                  {
//                      enabledSlots[index++] = slotIndex + 1;
//                  }
//              }

//              if (analyzeSpecificSlot && !specificSlotEnabled) // Error
//              {
//                  Console.WriteLine("ERROR: Specified analysis slot {0} is not enabled for waveform
{1}.", calDatum[calIndex].WaveConfig.AnalysisSlot, aiqFile);
//              }

//              double triggerLocation = 0;
//              double[] offsets = new double[numOfTriggers];

//              if (numOfTriggers == 0)
//              {
//                  Console.WriteLine("ERROR: No valid trigger available for waveform {0}.", aiqFile);
//              }
//              if (numOfTriggers == 1 && calDatum[calIndex].WaveConfig.Marker == 0)
//              {
//                  calDatum[calIndex].WaveConfig.MeasOffset = 0; // Assumes extermal trigger occurs
at enabled slot location
//              }
//              else if (numOfTriggers == 1)
//              {
//                  if (calDatum[calIndex].WaveConfig.Marker == 1) triggerLocation =
LteInfo.Marker.Marker1Info[0].OnTime * (frameLength / Ts);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 2) triggerLocation =
LteInfo.Marker.Marker2Info[0].OnTime * (frameLength / Ts);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 3) triggerLocation =
LteInfo.Marker.Marker3Info[0].OnTime * (frameLength / Ts);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 4) triggerLocation =
LteInfo.Marker.Marker4Info[0].OnTime * (frameLength / Ts);

//                  if (analyzeSpecificSlot)
//                  {
//                      offsets[0] = slotLocation[calDatum[calIndex].WaveConfig.AnalysisSlot - 1] -
triggerLocation;
//                      if (offsets[0] < 0) offsets[0] += frameLength; // TODO: Need to check if negative
offsets are valid
```

```
//                 }
//                 else
//                 {
//                     bool slotFound = false;

//                     // Find offset from trigger location to first enabled slot
//                     for (int slotIndex = 0; slotIndex < enabledSlots.Length; slotIndex++)
//                     {
//                         if (slotLocation[enabledSlots[slotIndex] - 1] >= triggerLocation)
//                         {
//                             offsets[0] = slotLocation[enabledSlots[slotIndex] - 1] - triggerLocation;
//                             slotFound = true;
//                         }

//                         if (slotFound) break;
//                     }

//                     // Find offset from trigger location to first enabled slot preceeding the trigger (uncommon)
//                     if (!slotFound)
//                     {
//                         offsets[0] = (slotLocation[enabledSlots[0] - 1] - triggerLocation) + frameLength;
//                         slotFound = true;
//                     }
//                 }

//                 calDatum[calIndex].WaveConfig.MeasOffset = offsets[0];
//             }
//             else if (numOfTriggers == numOfSlotsEnabled) // Assumes each trigger marks the location of an enabled slot
//             {
//                 // Calculate trigger offsets
//                 for (int slotIndex = 0; slotIndex < enabledSlots.Length; slotIndex++)
//                 {
//                     if (calDatum[calIndex].WaveConfig.Marker == 1) offsets[slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] - (LteInfo.Marker.Marker1Info[slotIndex].OnTime * (frameLength / Ts));
//                     else if (calDatum[calIndex].WaveConfig.Marker == 2) offsets[slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] - (LteInfo.Marker.Marker2Info[slotIndex].OnTime * (frameLength / Ts));
//                     else if (calDatum[calIndex].WaveConfig.Marker == 3) offsets[slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] - (LteInfo.Marker.Marker3Info[slotIndex].OnTime *
```

```
(frameLength / Ts));
//                  else if (calDatum[calIndex].WaveConfig.Marker == 4) offsets[slotIndex] =
slotLocation[enabledSlots[slotIndex] - 1] - (LteInfo.Marker.Marker4Info[slotIndex].OnTime *
(frameLength / Ts));
//              }

//              // Remove unnecessary digits from small double-precision value
//              double[] tmpOffsets = new double[offsets.Length];
//              for (int i = 0; i < offsets.Length; i++)
//              {
//                  tmpOffsets[i] = Math.Round(offsets[i], 12); // Round to picoseconds
//              }

//              // Check if trigger offsets are consistent
//              bool offsetMismatch = false;
//              if (tmpOffsets.Length > 1)
//              {
//                  for (int i = 1; i < tmpOffsets.Length; i++)
//                  {
//                      if (tmpOffsets[i] != tmpOffsets[i - 1]) offsetMismatch = true;
//                  }
//              }

//              // All trigger offsets are uniform, so set MeasOffset to any offset value
//              if (!offsetMismatch)
//              {
//                  calDatum[calIndex].WaveConfig.MeasOffset = tmpOffsets[0];
//              }
//              else
//              {
//                  // Error
//              }
//          }
//          else if (numOfTriggers < numOfSlotsEnabled)
//          {
//              double[,] offsetsAll = new double[numOfTriggers, numOfSlotsEnabled];

//              // Calculate all offsets from trigger location to each enabled slot
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
```

```
//                      if (calDatum[calIndex].WaveConfig.Marker == 1) offsetsAll[trigIndex, slotIndex]
= slotLocation[enabledSlots[slotIndex] - 1] - LteInfo.Marker.Marker1Info[trigIndex].OnTime *
(frameLength / Ts);
//                      else if (calDatum[calIndex].WaveConfig.Marker == 2) offsetsAll[trigIndex,
slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] -
LteInfo.Marker.Marker2Info[trigIndex].OnTime * (frameLength / Ts);
//                      else if (calDatum[calIndex].WaveConfig.Marker == 3) offsetsAll[trigIndex,
slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] -
LteInfo.Marker.Marker3Info[trigIndex].OnTime * (frameLength / Ts);
//                      else if (calDatum[calIndex].WaveConfig.Marker == 4) offsetsAll[trigIndex,
slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] -
LteInfo.Marker.Marker4Info[trigIndex].OnTime * (frameLength / Ts);

//                      offsetsAll[trigIndex, slotIndex] = Math.Round(offsetsAll[trigIndex, slotIndex], 9);
// Round to picoseconds
//                  }
//              }

//              // Find first valid (non-negative) offset for each trigger pulse
//              int[] offsetPosition = new int[numOfTriggers];
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (offsetsAll[trigIndex, slotIndex] >= 0)
//                      {
//                          offsetPosition[trigIndex] = slotIndex;
//                          break;
//                      }
//                  }
//              }

//              // Arrange/align offsets starting with valid (non-negative) values
//              double[,] tmpOffsetsAllAligned = new double[numOfTriggers, numOfSlotsEnabled];
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  int slotPosition = offsetPosition[trigIndex];

//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (slotPosition >= numOfSlotsEnabled) slotPosition = 0;
//                      tmpOffsetsAllAligned[trigIndex, slotIndex] = offsetsAll[trigIndex, slotPosition];
```

```
//                      slotPosition++;
//                  }
//              }

//              // Convert negative offsets to positive values
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (tmpOffsetsAllAligned[trigIndex, slotIndex] < 0)
tmpOffsetsAllAligned[trigIndex, slotIndex] += frameLength;
//                  }
//              }

//              // Remove unnecessary digits from small double-precision value
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      tmpOffsetsAllAligned[trigIndex, slotIndex] =
Math.Round(tmpOffsetsAllAligned[trigIndex, slotIndex], 9); // Round to picoseconds
//                  }
//              }

//              // Store trigger offsets for enabled slots within trigger(n) and trigger(n+1)
//              double trigPeriod = 0;
//              double trigPeriodMax = frameLength;
//              for (int trigIndex = 1; trigIndex <= numOfTriggers; trigIndex++)
//              {
//                  if (trigIndex < numOfTriggers) // Calculate amount of time between adjacent
triggers
//                  {
//                      if (calDatum[calIndex].WaveConfig.Marker == 1) trigPeriod =
(LteInfo.Marker.Marker1Info[trigIndex].OnTime - LteInfo.Marker.Marker1Info[trigIndex - 1].OnTime)
* (frameLength / Ts);
//                      else if (calDatum[calIndex].WaveConfig.Marker == 2) trigPeriod =
(LteInfo.Marker.Marker2Info[trigIndex].OnTime - LteInfo.Marker.Marker2Info[trigIndex - 1].OnTime)
* (frameLength / Ts);
//                      else if (calDatum[calIndex].WaveConfig.Marker == 3) trigPeriod =
(LteInfo.Marker.Marker3Info[trigIndex].OnTime - LteInfo.Marker.Marker3Info[trigIndex - 1].OnTime)
* (frameLength / Ts);
//                      else if (calDatum[calIndex].WaveConfig.Marker == 4) trigPeriod =
```

```
(LteInfo.Marker.Marker4Info[trigIndex].OnTime - LteInfo.Marker.Marker4Info[trigIndex - 1].OnTime)
* (frameLength / Ts);
//                  }
//                  else if (trigIndex == numOfTriggers) // Calculate amount of time between last
trigger and first trigger (waveform wrap-around)
//                  {
//                      if (calDatum[calIndex].WaveConfig.Marker == 1) trigPeriod = frameLength -
(LteInfo.Marker.Marker1Info[trigIndex - 1].OnTime * (frameLength / Ts)) +
(LteInfo.Marker.Marker1Info[0].OnTime * (frameLength / Ts));
//                      else if (calDatum[calIndex].WaveConfig.Marker == 2) trigPeriod = frameLength
- (LteInfo.Marker.Marker2Info[trigIndex - 1].OnTime * (frameLength / Ts)) +
(LteInfo.Marker.Marker2Info[0].OnTime * (frameLength / Ts));
//                      else if (calDatum[calIndex].WaveConfig.Marker == 3) trigPeriod = frameLength
- (LteInfo.Marker.Marker3Info[trigIndex - 1].OnTime * (frameLength / Ts)) +
(LteInfo.Marker.Marker3Info[0].OnTime * (frameLength / Ts));
//                      else if (calDatum[calIndex].WaveConfig.Marker == 4) trigPeriod = frameLength
- (LteInfo.Marker.Marker4Info[trigIndex - 1].OnTime * (frameLength / Ts)) +
(LteInfo.Marker.Marker4Info[0].OnTime * (frameLength / Ts));
//                  }

//                  trigPeriod = Math.Round(trigPeriod, 9); // Round to picoseconds
//                  if (trigPeriod < trigPeriodMax) trigPeriodMax = trigPeriod;
//              }

//              // Ignore offsets that are beyond the next consecutive trigger
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (tmpOffsetsAllAligned[trigIndex, slotIndex] >= trigPeriodMax)
tmpOffsetsAllAligned[trigIndex, slotIndex] = -9999;
//                  }
//              }

//              // Check if trigger offsets are consistent
//              bool offsetMismatch = false;
//              if (tmpOffsetsAllAligned.GetLength(0) > 1)
//              {
//                  for (int trigIndex = 1; trigIndex < tmpOffsetsAllAligned.GetLength(0); trigIndex++)
//                  {
//                      for (int slotIndex = 0; slotIndex < tmpOffsetsAllAligned.GetLength(1);
slotIndex++)
```

```
//                      {
//                          if (tmpOffsetsAllAligned[trigIndex, slotIndex] != -9999 &&
tmpOffsetsAllAligned[trigIndex, slotIndex] != tmpOffsetsAllAligned[trigIndex - 1, slotIndex])
offsetMismatch = true;
//                      }
//                  }
//              }

//              // All trigger offsets are the same, so set MeasOffset to first enabled slot
//              if (!offsetMismatch)
//              {
//                  calDatum[calIndex].WaveConfig.MeasOffset = tmpOffsetsAllAligned[0, 0];
//              }
//              else
//              {
//                  // Error
//              }
//          }
//          else if (numOfTriggers > numOfSlotsEnabled)
//          {
//              // ???
//          }

//          calDatum[calIndex].WaveConfig.WaveformFile = aiqFile;
//          calDatum[calIndex].WaveConfig.Bandwidth = LteInfo.Bandwidth;
//          calDatum[calIndex].WaveConfig.FrameLength = frameLength; // 10e-3;
//          calDatum[calIndex].WaveConfig.DataLength = slotLength;   // 0.5e-3
//          calDatum[calIndex].WaveConfig.MeasLength =
calDatum[calIndex].WaveConfig.MeasOffset + calDatum[calIndex].WaveConfig.DataLength;
//          calDatum[calIndex].WaveConfig.DutyCycle = (slotLength * numOfSlotsEnabled /
frameLength) * 100;

//          GetModulationSamplingParams(calDatum[calIndex].WaveConfig.Modulation,
calDatum[calIndex].srcFreq, calDatum[calIndex].WaveConfig.Bandwidth, calIndex, out double
minSampleFreq, out double recSampleFreq, out double measSpan);

//          calDatum[calIndex].WaveConfig.MinSampleFreq = minSampleFreq;
//          calDatum[calIndex].WaveConfig.RecSampleFreq = recSampleFreq;
//          calDatum[calIndex].WaveConfig.MeasSpan = measSpan;
//          calDatum[calIndex].WaveConfig.Headroom = 10;
//      }
//      catch (FileNotFoundException error)
```

```
//          {
//              Console.WriteLine("\n{0}", error.Message);
//              return -1;
//          }
//      }
//      else if (calDatum[calIndex].modulationType == "TD-SCDMA")
//      {
//          try
//          {
//              string aiqFile = calDatum[calIndex].modulationFile;
//              string iqsFile = waveDirectory + "Waveforms\\" +
Path.GetFileNameWithoutExtension(aiqFile) + ".iqs";

//              // Check if .iqs file exists for corresponding .aiq file
//              status = CheckWaveSettingsFile(iqsFile);

//              //////// Need to put an error here if file doesn'e exist ////////

//              // Get waveform data
//              Waveform TdscdmaWaveform = new Waveform();
//              TdscdmaWaveformInfo TdscdmaInfo =
TdscdmaWaveform.GetTdscdmaWaveformInfo(iqsFile);

//              const double chips = 6400;
//              const ushort maxSlots = 7;
//              //const double slotLength = 675e-6;
//              const double dataLength = 662.5e-6; // Data (275us) + Midamble (112.5us) + Data
(275us)
//              const double DwPTS = 75e-6;
//              const double UpPTS = 125e-6;
//              const double frameLength = 5e-3;

//              //double[] slotLocation = new double[maxSlots];
//              //slotLocation[0] = 0;        // TS0
//              //slotLocation[1] = 0.000950; // TS1
//              //slotLocation[2] = 0.001625; // TS2
//              //slotLocation[3] = 0.002300; // TS3
//              //slotLocation[4] = 0.002975; // TS4
//              //slotLocation[5] = 0.003650; // TS5
//              //slotLocation[6] = 0.004325; // TS6

//              double[] dataLocation = new double[maxSlots];
```

```
//              dataLocation[0] = 0;        // TS0
//              dataLocation[1] = 0.000950; // TS1
//              dataLocation[2] = 0.001625; // TS2
//              dataLocation[3] = 0.002300; // TS3
//              dataLocation[4] = 0.002975; // TS4
//              dataLocation[5] = 0.003650; // TS5
//              dataLocation[6] = 0.004325; // TS6

//              // Check whether specified waveform marker is valid
//              if (calDatum[calIndex].WaveConfig.Marker >= 1 &&
calDatum[calIndex].WaveConfig.Marker <= 4) // Marker 1 - 4
//              {
//                  if ((calDatum[calIndex].WaveConfig.Marker == 1 &&
TdscdmaInfo.Marker.Marker1Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 2 &&
TdscdmaInfo.Marker.Marker2Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 3 &&
TdscdmaInfo.Marker.Marker3Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 4 &&
TdscdmaInfo.Marker.Marker4Info.Count == 0))
//                  {
//                      Console.WriteLine("ERROR: Specified waveform marker {0} is undefined for
waveform {1}.", calDatum[calIndex].WaveConfig.Marker, aiqFile);
//                  }
//              }
//              else if (calDatum[calIndex].WaveConfig.Marker == -1) // No marker specified; default to
1st available marker
//              {
//                  if (TdscdmaInfo.Marker.Marker1Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 1;
//                  else if (TdscdmaInfo.Marker.Marker2Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 2;
//                  else if (TdscdmaInfo.Marker.Marker3Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 3;
//                  else if (TdscdmaInfo.Marker.Marker4Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 4;
//                  else // Error
//                  {
//                      Console.WriteLine("ERROR: No waveform marker(s) defined for waveform {0}.",
aiqFile);
//                  }
//              }
//              else if (calDatum[calIndex].WaveConfig.Marker != 0) // Marker 0 = external marker
//              {
//                  Console.WriteLine("ERROR: Invalid waveform marker {0} for waveform {1}.",
```

```
calDatum[calIndex].WaveConfig.Marker, aiqFile);
//              }

//              // Determine number of triggers for specified marker
//              int numOfTriggers = 0;
//              if (calDatum[calIndex].WaveConfig.Marker == 0) numOfTriggers = 1;
//              else if (calDatum[calIndex].WaveConfig.Marker == 1) numOfTriggers =
TdscdmaInfo.Marker.Marker1Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 2) numOfTriggers =
TdscdmaInfo.Marker.Marker2Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 3) numOfTriggers =
TdscdmaInfo.Marker.Marker3Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 4) numOfTriggers =
TdscdmaInfo.Marker.Marker4Info.Count;

//              // Check whether to use specific slot for capture/analysis
//              bool analyzeSpecificSlot = false;
//              if (calDatum[calIndex].WaveConfig.AnalysisSlot != -1)
//              {
//                  if (calDatum[calIndex].WaveConfig.AnalysisSlot >= 1 &&
calDatum[calIndex].WaveConfig.AnalysisSlot <= maxSlots)
//                  {
//                      analyzeSpecificSlot = true;
//                  }
//                  else // Error
//                  {
//                      Console.WriteLine("Requested analysis slot {0} out of range for TD-SCDMA (valid
slot is 1 - {1}).", calDatum[calIndex].WaveConfig.AnalysisSlot, maxSlots);
//                  }
//              }

//              // Determine how many TSx slots are enabled
//              int numOfSlotsEnabled = 0;
//              bool specificSlotEnabled = false;
//              for (int slot = 0; slot < maxSlots; slot++)
//              {
//                  if (TdscdmaInfo.SlotEnabled[slot] == true)
//                  {
//                      numOfSlotsEnabled++;
//                      if (slot == calDatum[calIndex].WaveConfig.AnalysisSlot) specificSlotEnabled =
true;
//                  }
```

```
//              }

//              if (numOfSlotsEnabled < 1) // Error
//              {
//                  Console.WriteLine("ERROR: No slots enabled for waveform {0}.", aiqFile);
//              }

//              // Determine if DwPTS On slot is enabled
//              //bool DwPtsSlotsEnabled = false;
//              //if (TdscdmaInfo.DwPtsEnabled) DwPtsSlotsEnabled = true;

//              // Determine if UpPTS On slot is enabled
//              //bool UpPtsSlotsEnabled = false;
//              //if (TdscdmaInfo.UpPtsEnabled) UpPtsSlotsEnabled = true;

//              // Generate list of enabled slots
//              int index = 0;
//              int[] enabledSlots = new int[numOfSlotsEnabled];
//              for (int slotIndex = 0; slotIndex < maxSlots; slotIndex++)
//              {
//                  if (TdscdmaInfo.SlotEnabled[slotIndex] == true)
//                  {
//                      enabledSlots[index++] = slotIndex + 1;
//                  }
//              }

//              if (analyzeSpecificSlot && !specificSlotEnabled) // Error
//              {
//                  Console.WriteLine("ERROR: Specified analysis slot {0} is not enabled for waveform
{1}.", calDatum[calIndex].WaveConfig.AnalysisSlot, aiqFile);
//              }

//              double triggerLocation = 0;
//              double[] offsets = new double[numOfTriggers];

//              if (numOfTriggers == 0)
//              {
//                  Console.WriteLine("ERROR: No valid trigger available for waveform {0}.", aiqFile);
//              }
//              if (numOfTriggers == 1 && calDatum[calIndex].WaveConfig.Marker == 0)
//              {
//                  calDatum[calIndex].WaveConfig.MeasOffset = 0; // Assumes extermal trigger occurs
```

```
at enabled slot location
//              }
//              else if (numOfTriggers == 1)
//              {
//                  if (calDatum[calIndex].WaveConfig.Marker == 1) triggerLocation =
TdscdmaInfo.Marker.Marker1Info[0].OnTime * (frameLength / chips);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 2) triggerLocation =
TdscdmaInfo.Marker.Marker2Info[0].OnTime * (frameLength / chips);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 3) triggerLocation =
TdscdmaInfo.Marker.Marker3Info[0].OnTime * (frameLength / chips);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 4) triggerLocation =
TdscdmaInfo.Marker.Marker4Info[0].OnTime * (frameLength / chips);

//                  if (analyzeSpecificSlot)
//                  {
//                      offsets[0] = dataLocation[calDatum[calIndex].WaveConfig.AnalysisSlot] -
triggerLocation;
//                      if (offsets[0] < 0) offsets[0] += frameLength; // TODO: Need to check if negative
offsets are valid
//                  }
//                  else
//                  {
//                      bool slotFound = false;

//                      // Find offset from trigger location to first enabled slot
//                      for (int slotIndex = 0; slotIndex < enabledSlots.Length; slotIndex++)
//                      {
//                          if (dataLocation[enabledSlots[slotIndex] - 1] >= triggerLocation)
//                          {
//                              offsets[0] = dataLocation[enabledSlots[slotIndex] - 1] - triggerLocation;
//                              slotFound = true;
//                          }

//                          if (slotFound) break;
//                      }

//                      // Find offset from trigger location to first enabled slot preceeding the trigger
(uncommon)
//                      if (!slotFound)
//                      {
//                          offsets[0] = (dataLocation[enabledSlots[0]] - triggerLocation) + frameLength;
//                          slotFound = true;
```

```
//                  }
//              }

//              calDatum[calIndex].WaveConfig.MeasOffset = offsets[0];
//          }
//          else if (numOfTriggers == numOfSlotsEnabled) // Assumes each trigger marks the
location of an enabled slot
//          {
//              // Calculate trigger offsets
//              for (int slotIndex = 0; slotIndex < enabledSlots.Length; slotIndex++)
//              {
//                  if (calDatum[calIndex].WaveConfig.Marker == 1) offsets[slotIndex] =
dataLocation[enabledSlots[slotIndex]] - (TdscdmaInfo.Marker.Marker1Info[slotIndex].OnTime *
(frameLength / chips));
//                  else if (calDatum[calIndex].WaveConfig.Marker == 2) offsets[slotIndex] =
dataLocation[enabledSlots[slotIndex]] - (TdscdmaInfo.Marker.Marker2Info[slotIndex].OnTime *
(frameLength / chips));
//                  else if (calDatum[calIndex].WaveConfig.Marker == 3) offsets[slotIndex] =
dataLocation[enabledSlots[slotIndex]] - (TdscdmaInfo.Marker.Marker3Info[slotIndex].OnTime *
(frameLength / chips));
//                  else if (calDatum[calIndex].WaveConfig.Marker == 4) offsets[slotIndex] =
dataLocation[enabledSlots[slotIndex]] - (TdscdmaInfo.Marker.Marker4Info[slotIndex].OnTime *
(frameLength / chips));
//                  index++;
//              }

//              // Remove unnecessary digits from small double-precision value
//              double[] tmpOffsets = new double[offsets.Length];
//              for (int i = 0; i < offsets.Length; i++)
//              {
//                  tmpOffsets[i] = Math.Round(offsets[i], 12); // Round to picoseconds
//              }

//              // Check if trigger offsets are consistent
//              bool offsetMismatch = false;
//              if (tmpOffsets.Length > 1)
//              {
//                  for (int i = 1; i < tmpOffsets.Length; i++)
//                  {
//                      if (tmpOffsets[i] != tmpOffsets[i - 1]) offsetMismatch = true;
//                  }
//              }
```

```
//                  // All trigger offsets are uniform, so set MeasOffset to any offset value
//                  if (!offsetMismatch)
//                  {
//                      calDatum[calIndex].WaveConfig.MeasOffset = tmpOffsets[0];
//                  }
//                  else
//                  {
//                      // Error
//                  }
//              }
//              else if (numOfTriggers < numOfSlotsEnabled)
//              {
//                  double[,] offsetsAll = new double[numOfTriggers, numOfSlotsEnabled];

//                  // Calculate all offsets from trigger location to each enabled slot
//                  for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//                  {
//                      for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                      {
//                          if (calDatum[calIndex].WaveConfig.Marker == 1) offsetsAll[trigIndex, slotIndex]
= dataLocation[enabledSlots[slotIndex] - 1] - (TdscdmaInfo.Marker.Marker1Info[trigIndex].OnTime
* (frameLength / chips));
//                          else if (calDatum[calIndex].WaveConfig.Marker == 2) offsetsAll[trigIndex,
slotIndex] = dataLocation[enabledSlots[slotIndex] - 1] -
(TdscdmaInfo.Marker.Marker2Info[trigIndex].OnTime * (frameLength / chips));
//                          else if (calDatum[calIndex].WaveConfig.Marker == 3) offsetsAll[trigIndex,
slotIndex] = dataLocation[enabledSlots[slotIndex] - 1] -
(TdscdmaInfo.Marker.Marker3Info[trigIndex].OnTime * (frameLength / chips));
//                          else if (calDatum[calIndex].WaveConfig.Marker == 4) offsetsAll[trigIndex,
slotIndex] = dataLocation[enabledSlots[slotIndex] - 1] -
(TdscdmaInfo.Marker.Marker4Info[trigIndex].OnTime * (frameLength / chips));

//                          offsetsAll[trigIndex, slotIndex] = Math.Round(offsetsAll[trigIndex, slotIndex], 9);
// Round to picoseconds
//                      }
//                  }

//                  // Find first valid (non-negative) offset for each trigger pulse
//                  int[] offsetPosition = new int[numOfTriggers];
//                  for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//                  {
```

```
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (offsetsAll[trigIndex, slotIndex] >= 0)
//                      {
//                          offsetPosition[trigIndex] = slotIndex;
//                          break;
//                      }
//                  }
//              }

//              // Arrange/align offsets starting with valid (non-negative) values
//              double[,] tmpOffsetsAllAligned = new double[numOfTriggers, numOfSlotsEnabled];
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  int slotPosition = offsetPosition[trigIndex];

//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (slotPosition >= numOfSlotsEnabled) slotPosition = 0;
//                      tmpOffsetsAllAligned[trigIndex, slotIndex] = offsetsAll[trigIndex, slotPosition];
//                      slotPosition++;
//                  }
//              }

//              // Convert negative offsets to positive values
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (tmpOffsetsAllAligned[trigIndex, slotIndex] < 0)
tmpOffsetsAllAligned[trigIndex, slotIndex] += frameLength;
//                  }
//              }

//              // Remove unnecessary digits from small double-precision value
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      tmpOffsetsAllAligned[trigIndex, slotIndex] =
Math.Round(tmpOffsetsAllAligned[trigIndex, slotIndex], 9); // Round to picoseconds
//                  }
```

```
//                  }

//                  // Store trigger offsets for enabled slots within trigger(n) and trigger(n+1)
//                  double trigPeriod = 0;
//                  double trigPeriodMax = frameLength;
//                  for (int trigIndex = 1; trigIndex <= numOfTriggers; trigIndex++)
//                  {
//                      if (trigIndex < numOfTriggers) // Calculate amount of time between adjacent
triggers
//                      {
//                          if (calDatum[calIndex].WaveConfig.Marker == 1) trigPeriod =
(TdscdmaInfo.Marker.Marker1Info[trigIndex].OnTime - TdscdmaInfo.Marker.Marker1Info[trigIndex
- 1].OnTime) * (frameLength / chips);
//                          else if (calDatum[calIndex].WaveConfig.Marker == 2) trigPeriod =
(TdscdmaInfo.Marker.Marker2Info[trigIndex].OnTime - TdscdmaInfo.Marker.Marker2Info[trigIndex
- 1].OnTime) * (frameLength / chips);
//                          else if (calDatum[calIndex].WaveConfig.Marker == 3) trigPeriod =
(TdscdmaInfo.Marker.Marker3Info[trigIndex].OnTime - TdscdmaInfo.Marker.Marker3Info[trigIndex
- 1].OnTime) * (frameLength / chips);
//                          else if (calDatum[calIndex].WaveConfig.Marker == 4) trigPeriod =
(TdscdmaInfo.Marker.Marker4Info[trigIndex].OnTime - TdscdmaInfo.Marker.Marker4Info[trigIndex
- 1].OnTime) * (frameLength / chips);
//                      }
//                      else if (trigIndex == numOfTriggers) // Calculate amount of time between last
trigger and first trigger (waveform wrap-around)
//                      {
//                          if (calDatum[calIndex].WaveConfig.Marker == 1) trigPeriod = frameLength -
(TdscdmaInfo.Marker.Marker1Info[trigIndex - 1].OnTime * (frameLength / chips)) +
(TdscdmaInfo.Marker.Marker1Info[0].OnTime * (frameLength / chips));
//                          else if (calDatum[calIndex].WaveConfig.Marker == 2) trigPeriod = frameLength
- (TdscdmaInfo.Marker.Marker2Info[trigIndex - 1].OnTime * (frameLength / chips)) +
(TdscdmaInfo.Marker.Marker2Info[0].OnTime * (frameLength / chips));
//                          else if (calDatum[calIndex].WaveConfig.Marker == 3) trigPeriod = frameLength
- (TdscdmaInfo.Marker.Marker3Info[trigIndex - 1].OnTime * (frameLength / chips)) +
(TdscdmaInfo.Marker.Marker3Info[0].OnTime * (frameLength / chips));
//                          else if (calDatum[calIndex].WaveConfig.Marker == 4) trigPeriod = frameLength
- (TdscdmaInfo.Marker.Marker4Info[trigIndex - 1].OnTime * (frameLength / chips)) +
(TdscdmaInfo.Marker.Marker4Info[0].OnTime * (frameLength / chips));
//                      }

//                      trigPeriod = Math.Round(trigPeriod, 9); // Round to picoseconds
//                      if (trigPeriod < trigPeriodMax) trigPeriodMax = trigPeriod;
```

```
//                }

//                // Ignore offsets that are beyond the next consecutive trigger
//                for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//                {
//                    for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                    {
//                        if (tmpOffsetsAllAligned[trigIndex, slotIndex] >= trigPeriodMax)
tmpOffsetsAllAligned[trigIndex, slotIndex] = -9999;
//                    }
//                }

//                // Check if trigger offsets are consistent
//                bool offsetMismatch = false;
//                if (tmpOffsetsAllAligned.GetLength(0) > 1)
//                {
//                    for (int trigIndex = 1; trigIndex < tmpOffsetsAllAligned.GetLength(0); trigIndex++)
//                    {
//                        for (int slotIndex = 0; slotIndex < tmpOffsetsAllAligned.GetLength(1);
slotIndex++)
//                        {
//                            if (tmpOffsetsAllAligned[trigIndex, slotIndex] != -9999 &&
tmpOffsetsAllAligned[trigIndex, slotIndex] != tmpOffsetsAllAligned[trigIndex - 1, slotIndex])
offsetMismatch = true;
//                        }
//                    }
//                }

//                // All trigger offsets are the same, so set MeasOffset to first enabled slot
//                if (!offsetMismatch)
//                {
//                    calDatum[calIndex].WaveConfig.MeasOffset = tmpOffsetsAllAligned[0, 0];
//                }
//                else
//                {
//                    // Error
//                }
//            }

//            calDatum[calIndex].WaveConfig.Modulation = ModulationType.TDSCDMA;
//            calDatum[calIndex].WaveConfig.WaveformFile = aiqFile;
//            calDatum[calIndex].WaveConfig.Bandwidth = ChannelBandwidth.NA; // 1.6 MHz
```

```
//              calDatum[calIndex].WaveConfig.FrameLength = frameLength;
//              calDatum[calIndex].WaveConfig.DataLength = dataLength;
//              calDatum[calIndex].WaveConfig.MeasLength =
calDatum[calIndex].WaveConfig.MeasOffset + calDatum[calIndex].WaveConfig.DataLength;

//              // Calculate duty cycle
//              double powerOnTime = dataLength * numOfSlotsEnabled;
//              if (TdscdmaInfo.DwPtsEnabled) powerOnTime += DwPTS;
//              if (TdscdmaInfo.UpPtsEnabled) powerOnTime += UpPTS;
//              calDatum[calIndex].WaveConfig.DutyCycle = (powerOnTime / frameLength) * 100;

//              GetModulationSamplingParams(calDatum[calIndex].WaveConfig.Modulation,
calDatum[calIndex].srcFreq, calDatum[calIndex].WaveConfig.Bandwidth, calIndex, out double
minSampleFreq, out double recSampleFreq, out double measSpan);

//              calDatum[calIndex].WaveConfig.MinSampleFreq = minSampleFreq;
//              calDatum[calIndex].WaveConfig.RecSampleFreq = recSampleFreq;
//              calDatum[calIndex].WaveConfig.MeasSpan = measSpan;
//              calDatum[calIndex].WaveConfig.Headroom = 8;
//          }
//          catch (FileNotFoundException error)
//          {
//              Console.WriteLine("\n{0}", error.Message);
//              return -1;
//          }
//      }
//      else if (calDatum[calIndex].modulationType == "WCDMA")
//      {
//          try
//          {
//              string aiqFile = calDatum[calIndex].modulationFile;
//              string iqsFile = waveDirectory + "Waveforms\\" +
Path.GetFileNameWithoutExtension(aiqFile) + ".iqs";

//              // Check if .iqs file exists for corresponding .aiq file
//              status = CheckWaveSettingsFile(iqsFile);

//              //////// Need to put an error here if file doesn'e exist ////////

//              // Get waveform data
//              Waveform WcdmaWaveform = new Waveform();
//              WcdmaWaveformInfo WcdmaInfo =
```

```
WcdmaWaveform.GetWcdmaWaveformInfo(iqsFile);

//              const uint chips = 38400;
//              const ushort maxSlots = 20;
//              const ushort maxSubframes = 10;
//              const double slotLength = 0.5e-3;
//              const double subFrameLength = 1e-3;
//              const double frameLength = 10e-3;

//              double[] slotLocation = new double[maxSlots];
//              slotLocation[0] = 0;
//              slotLocation[1] = 0.0005;
//              slotLocation[2] = 0.001;
//              slotLocation[3] = 0.0015;
//              slotLocation[4] = 0.002;
//              slotLocation[5] = 0.0025;
//              slotLocation[6] = 0.003;
//              slotLocation[7] = 0.0035;
//              slotLocation[8] = 0.004;
//              slotLocation[9] = 0.0045;
//              slotLocation[10] = 0.005;
//              slotLocation[11] = 0.0055;
//              slotLocation[12] = 0.006;
//              slotLocation[13] = 0.0065;
//              slotLocation[14] = 0.007;
//              slotLocation[15] = 0.0075;
//              slotLocation[16] = 0.008;
//              slotLocation[17] = 0.0085;
//              slotLocation[18] = 0.009;
//              slotLocation[19] = 0.0095;

//              // Check whether specified waveform marker is valid
//              if (calDatum[calIndex].WaveConfig.Marker >= 1 &&
calDatum[calIndex].WaveConfig.Marker <= 4) // Marker 1 - 4
//              {
//                  if ((calDatum[calIndex].WaveConfig.Marker == 1 &&
WcdmaInfo.Marker.Marker1Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 2 &&
WcdmaInfo.Marker.Marker2Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 3 &&
WcdmaInfo.Marker.Marker3Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 4 &&
WcdmaInfo.Marker.Marker4Info.Count == 0))
//                  {
//                      Console.WriteLine("ERROR: Specified waveform marker {0} is undefined for
```

```
waveform {1}.", calDatum[calIndex].WaveConfig.Marker, aiqFile);
//                 }
//             }
//             else if (calDatum[calIndex].WaveConfig.Marker == -1) // No marker specified; default to
1st available marker
//             {
//                 if (WcdmaInfo.Marker.Marker1Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 1;
//                 else if (WcdmaInfo.Marker.Marker2Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 2;
//                 else if (WcdmaInfo.Marker.Marker3Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 3;
//                 else if (WcdmaInfo.Marker.Marker4Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 4;
//                 else // Error
//                 {
//                     Console.WriteLine("ERROR: No waveform marker(s) defined for waveform {0}.",
aiqFile);
//                 }
//             }
//             else if (calDatum[calIndex].WaveConfig.Marker != 0) // Marker 0 = external marker
//             {
//                 Console.WriteLine("ERROR: Invalid waveform marker {0} for waveform {1}.",
calDatum[calIndex].WaveConfig.Marker, aiqFile);
//             }

//             // Determine number of triggers for specified marker
//             int numOfTriggers = 0;
//             if (calDatum[calIndex].WaveConfig.Marker == 0) numOfTriggers = 1;
//             else if (calDatum[calIndex].WaveConfig.Marker == 1) numOfTriggers =
WcdmaInfo.Marker.Marker1Info.Count;
//             else if (calDatum[calIndex].WaveConfig.Marker == 2) numOfTriggers =
WcdmaInfo.Marker.Marker2Info.Count;
//             else if (calDatum[calIndex].WaveConfig.Marker == 3) numOfTriggers =
WcdmaInfo.Marker.Marker3Info.Count;
//             else if (calDatum[calIndex].WaveConfig.Marker == 4) numOfTriggers =
WcdmaInfo.Marker.Marker4Info.Count;

//             // Check whether to use specific slot for capture/analysis
//             bool analyzeSpecificSlot = false;
//             if (calDatum[calIndex].WaveConfig.AnalysisSlot != -1)
//             {
```

```
//                  if (calDatum[calIndex].WaveConfig.AnalysisSlot >= 1 &&
calDatum[calIndex].WaveConfig.AnalysisSlot <= maxSlots)
//                  {
//                      analyzeSpecificSlot = true;
//                  }
//                  else // Error
//                  {
//                      Console.WriteLine("Requested analysis slot {0} out of range for LTE (valid slot is
1 - {1}).", calDatum[calIndex].WaveConfig.AnalysisSlot, maxSlots);
//                  }
//              }

//              // Determine how many slots are enabled
//              int numOfSlotsEnabled = 0;
//              bool specificSlotEnabled = false;
//              for (int slot = 0; slot < maxSlots; slot++)
//              {
//                  if (WcdmaInfo.SlotEnabled[slot] == true)
//                  {
//                      numOfSlotsEnabled++;
//                      if (slot + 1 == calDatum[calIndex].WaveConfig.AnalysisSlot) specificSlotEnabled =
true;
//                  }
//              }

//              if (numOfSlotsEnabled < 1) // Error
//              {
//                  Console.WriteLine("ERROR: No slots enabled for waveform {0}.", aiqFile);
//              }

//              // Generate list of enabled slots
//              int index = 0;
//              int[] enabledSlots = new int[numOfSlotsEnabled];
//              for (int slotIndex = 0; slotIndex < maxSlots; slotIndex++)
//              {
//                  if (WcdmaInfo.SlotEnabled[slotIndex] == true)
//                  {
//                      enabledSlots[index++] = slotIndex + 1;
//                  }
//              }

//              if (analyzeSpecificSlot && !specificSlotEnabled) // Error
```

```
//              {
//                  Console.WriteLine("ERROR: Specified analysis slot {0} is not enabled for waveform
{1}.", calDatum[calIndex].WaveConfig.AnalysisSlot, aiqFile);
//              }

//              double triggerLocation = 0;
//              double[] offsets = new double[numOfTriggers];

//              if (numOfTriggers == 0)
//              {
//                  Console.WriteLine("ERROR: No valid trigger available for waveform {0}.", aiqFile);
//              }
//              if (numOfTriggers == 1 && calDatum[calIndex].WaveConfig.Marker == 0)
//              {
//                  calDatum[calIndex].WaveConfig.MeasOffset = 0; // Assumes extermal trigger occurs
at enabled slot location
//              }
//              else if (numOfTriggers == 1)
//              {
//                  if (calDatum[calIndex].WaveConfig.Marker == 1) triggerLocation =
WcdmaInfo.Marker.Marker1Info[0].OnTime * (frameLength / chips);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 2) triggerLocation =
WcdmaInfo.Marker.Marker2Info[0].OnTime * (frameLength / chips);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 3) triggerLocation =
WcdmaInfo.Marker.Marker3Info[0].OnTime * (frameLength / chips);
//                  else if (calDatum[calIndex].WaveConfig.Marker == 4) triggerLocation =
WcdmaInfo.Marker.Marker4Info[0].OnTime * (frameLength / chips);

//                  if (analyzeSpecificSlot)
//                  {
//                      offsets[0] = slotLocation[calDatum[calIndex].WaveConfig.AnalysisSlot - 1] -
triggerLocation;
//                      if (offsets[0] < 0) offsets[0] += frameLength; // TODO: Need to check if negative
offsets are valid
//                  }
//                  else
//                  {
//                      bool slotFound = false;

//                      // Find offset from trigger location to first enabled slot
//                      for (int slotIndex = 0; slotIndex < enabledSlots.Length; slotIndex++)
//                      {
```

```
//                      if (slotLocation[enabledSlots[slotIndex] - 1] >= triggerLocation)
//                      {
//                          offsets[0] = slotLocation[enabledSlots[slotIndex] - 1] - triggerLocation;
//                          slotFound = true;
//                      }

//                      if (slotFound) break;
//                  }

//                  // Find offset from trigger location to first enabled slot preceeding the trigger
(uncommon)
//                  if (!slotFound)
//                  {
//                      offsets[0] = (slotLocation[enabledSlots[0] - 1] - triggerLocation) + frameLength;
//                      slotFound = true;
//                  }
//              }

//                  calDatum[calIndex].WaveConfig.MeasOffset = offsets[0];
//              }
//              else if (numOfTriggers == numOfSlotsEnabled) // Assumes each trigger marks the
location of an enabled slot
//              {
//                  // Calculate trigger offsets
//                  for (int slotIndex = 0; slotIndex < enabledSlots.Length; slotIndex++)
//                  {
//                      if (calDatum[calIndex].WaveConfig.Marker == 1) offsets[slotIndex] =
slotLocation[enabledSlots[slotIndex] - 1] - (WcdmaInfo.Marker.Marker1Info[slotIndex].OnTime *
(frameLength / chips));
//                      if (calDatum[calIndex].WaveConfig.Marker == 2) offsets[slotIndex] =
slotLocation[enabledSlots[slotIndex] - 1] - (WcdmaInfo.Marker.Marker2Info[slotIndex].OnTime *
(frameLength / chips));
//                      if (calDatum[calIndex].WaveConfig.Marker == 3) offsets[slotIndex] =
slotLocation[enabledSlots[slotIndex] - 1] - (WcdmaInfo.Marker.Marker3Info[slotIndex].OnTime *
(frameLength / chips));
//                      if (calDatum[calIndex].WaveConfig.Marker == 4) offsets[slotIndex] =
slotLocation[enabledSlots[slotIndex] - 1] - (WcdmaInfo.Marker.Marker4Info[slotIndex].OnTime *
(frameLength / chips));
//                  }

//                  // Remove unnecessary digits from small double-precision value
//                  double[] tmpOffsets = new double[offsets.Length];
```

```
//                 for (int i = 0; i < offsets.Length; i++)
//                 {
//                     tmpOffsets[i] = Math.Round(offsets[i], 12); // Round to picoseconds
//                 }

//                 // Check if trigger offsets are consistent
//                 bool offsetMismatch = false;
//                 if (tmpOffsets.Length > 1)
//                 {
//                     for (int i = 1; i < tmpOffsets.Length; i++)
//                     {
//                         if (tmpOffsets[i] != tmpOffsets[i - 1]) offsetMismatch = true;
//                     }
//                 }

//                 // All trigger offsets are uniform, so set MeasOffset to any offset value
//                 if (!offsetMismatch)
//                 {
//                     calDatum[calIndex].WaveConfig.MeasOffset = tmpOffsets[0];
//                 }
//                 else
//                 {
//                     // Error
//                 }
//             }
//             else if (numOfTriggers < numOfSlotsEnabled)
//             {
//                 double[,] offsetsAll = new double[numOfTriggers, numOfSlotsEnabled];

//                 // Calculate all offsets from trigger location to each enabled slot
//                 for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//                 {
//                     for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                     {
//                         if (calDatum[calIndex].WaveConfig.Marker == 1) offsetsAll[trigIndex, slotIndex]
= slotLocation[enabledSlots[slotIndex] - 1] - WcdmaInfo.Marker.Marker1Info[trigIndex].OnTime *
(frameLength / chips);
//                         else if (calDatum[calIndex].WaveConfig.Marker == 2) offsetsAll[trigIndex,
slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] -
WcdmaInfo.Marker.Marker2Info[trigIndex].OnTime * (frameLength / chips);
//                         else if (calDatum[calIndex].WaveConfig.Marker == 3) offsetsAll[trigIndex,
slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] -
```

```
WcdmaInfo.Marker.Marker3Info[trigIndex].OnTime * (frameLength / chips);
//                      else if (calDatum[calIndex].WaveConfig.Marker == 4) offsetsAll[trigIndex,
slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] -
WcdmaInfo.Marker.Marker4Info[trigIndex].OnTime * (frameLength / chips);

//                      offsetsAll[trigIndex, slotIndex] = Math.Round(offsetsAll[trigIndex, slotIndex], 9);
// Round to picoseconds
//                  }
//              }

//              // Find first valid (non-negative) offset for each trigger pulse
//              int[] offsetPosition = new int[numOfTriggers];
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (offsetsAll[trigIndex, slotIndex] >= 0)
//                      {
//                          offsetPosition[trigIndex] = slotIndex;
//                          break;
//                      }
//                  }
//              }

//              // Arrange/align offsets starting with valid (non-negative) values
//              double[,] tmpOffsetsAllAligned = new double[numOfTriggers, numOfSlotsEnabled];
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  int slotPosition = offsetPosition[trigIndex];

//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (slotPosition >= numOfSlotsEnabled) slotPosition = 0;
//                      tmpOffsetsAllAligned[trigIndex, slotIndex] = offsetsAll[trigIndex, slotPosition];
//                      slotPosition++;
//                  }
//              }

//              // Convert negative offsets to positive values
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
```

```
//                  {
//                      if (tmpOffsetsAllAligned[trigIndex, slotIndex] < 0)
tmpOffsetsAllAligned[trigIndex, slotIndex] += frameLength;
//                  }
//              }

//              // Remove unnecessary digits from small double-precision value
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      tmpOffsetsAllAligned[trigIndex, slotIndex] =
Math.Round(tmpOffsetsAllAligned[trigIndex, slotIndex], 9); // Round to picoseconds
//                  }
//              }

//              // Store trigger offsets for enabled slots within trigger(n) and trigger(n+1)
//              double trigPeriod = 0;
//              double trigPeriodMax = frameLength;
//              for (int trigIndex = 1; trigIndex <= numOfTriggers; trigIndex++)
//              {
//                  if (trigIndex < numOfTriggers) // Calculate amount of time between adjacent
triggers
//                  {
//                      if (calDatum[calIndex].WaveConfig.Marker == 1) trigPeriod =
(WcdmaInfo.Marker.Marker1Info[trigIndex].OnTime - WcdmaInfo.Marker.Marker1Info[trigIndex -
1].OnTime) * (frameLength / chips);
//                      else if (calDatum[calIndex].WaveConfig.Marker == 2) trigPeriod =
(WcdmaInfo.Marker.Marker2Info[trigIndex].OnTime - WcdmaInfo.Marker.Marker2Info[trigIndex -
1].OnTime) * (frameLength / chips);
//                      else if (calDatum[calIndex].WaveConfig.Marker == 3) trigPeriod =
(WcdmaInfo.Marker.Marker3Info[trigIndex].OnTime - WcdmaInfo.Marker.Marker3Info[trigIndex -
1].OnTime) * (frameLength / chips);
//                      else if (calDatum[calIndex].WaveConfig.Marker == 4) trigPeriod =
(WcdmaInfo.Marker.Marker4Info[trigIndex].OnTime - WcdmaInfo.Marker.Marker4Info[trigIndex -
1].OnTime) * (frameLength / chips);
//                  }
//                  else if (trigIndex == numOfTriggers) // Calculate amount of time between last
trigger and first trigger (waveform wrap-around)
//                  {
//                      if (calDatum[calIndex].WaveConfig.Marker == 1) trigPeriod = frameLength -
(WcdmaInfo.Marker.Marker1Info[trigIndex - 1].OnTime * (frameLength / chips)) +
```

```
(WcdmaInfo.Marker.Marker1Info[0].OnTime * (frameLength / chips));
//                      else if (calDatum[calIndex].WaveConfig.Marker == 2) trigPeriod = frameLength
- (WcdmaInfo.Marker.Marker2Info[trigIndex - 1].OnTime * (frameLength / chips)) +
(WcdmaInfo.Marker.Marker2Info[0].OnTime * (frameLength / chips));
//                      else if (calDatum[calIndex].WaveConfig.Marker == 3) trigPeriod = frameLength
- (WcdmaInfo.Marker.Marker3Info[trigIndex - 1].OnTime * (frameLength / chips)) +
(WcdmaInfo.Marker.Marker3Info[0].OnTime * (frameLength / chips));
//                      else if (calDatum[calIndex].WaveConfig.Marker == 4) trigPeriod = frameLength
- (WcdmaInfo.Marker.Marker4Info[trigIndex - 1].OnTime * (frameLength / chips)) +
(WcdmaInfo.Marker.Marker4Info[0].OnTime * (frameLength / chips));
//                  }

//                  trigPeriod = Math.Round(trigPeriod, 9); // Round to picoseconds
//                  if (trigPeriod < trigPeriodMax) trigPeriodMax = trigPeriod;
//              }

//              // Ignore offsets that are beyond the next consecutive trigger
//              for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              {
//                  for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  {
//                      if (tmpOffsetsAllAligned[trigIndex, slotIndex] >= trigPeriodMax)
tmpOffsetsAllAligned[trigIndex, slotIndex] = -9999;
//                  }
//              }

//              // Check if trigger offsets are consistent
//              bool offsetMismatch = false;
//              if (tmpOffsetsAllAligned.GetLength(0) > 1)
//              {
//                  for (int trigIndex = 1; trigIndex < tmpOffsetsAllAligned.GetLength(0); trigIndex++)
//                  {
//                      for (int slotIndex = 0; slotIndex < tmpOffsetsAllAligned.GetLength(1);
slotIndex++)
//                      {
//                          if (tmpOffsetsAllAligned[trigIndex, slotIndex] != -9999 &&
tmpOffsetsAllAligned[trigIndex, slotIndex] != tmpOffsetsAllAligned[trigIndex - 1, slotIndex])
offsetMismatch = true;
//                      }
//                  }
//              }
```

```
//                 // All trigger offsets are the same, so set MeasOffset to first enabled slot
//                 if (!offsetMismatch)
//                 {
//                     calDatum[calIndex].WaveConfig.MeasOffset = tmpOffsetsAllAligned[0, 0];
//                 }
//                 else
//                 {
//                     // Warning
//                 }
//             }
//             else if (numOfTriggers > numOfSlotsEnabled)
//             {
//                 // ???
//             }

//             calDatum[calIndex].WaveConfig.Modulation = ModulationType.WCDMA;
//             calDatum[calIndex].WaveConfig.WaveformFile = aiqFile;
//             calDatum[calIndex].WaveConfig.Bandwidth = ChannelBandwidth.NA; // 5 MHz
//             calDatum[calIndex].WaveConfig.FrameLength = frameLength; // 10e-3;
//             calDatum[calIndex].WaveConfig.DataLength = slotLength;   // 0.5e-3
//             calDatum[calIndex].WaveConfig.MeasLength =
calDatum[calIndex].WaveConfig.MeasOffset + calDatum[calIndex].WaveConfig.DataLength;
//             calDatum[calIndex].WaveConfig.DutyCycle = (slotLength * numOfSlotsEnabled /
frameLength) * 100;
//             calDatum[calIndex].WaveConfig.MinSampleFreq = 160e6;
//             calDatum[calIndex].WaveConfig.RecSampleFreq = 160e6;
//             calDatum[calIndex].WaveConfig.MeasSpan = 0;
//             calDatum[calIndex].WaveConfig.Headroom = 10;
//         }
//         catch (FileNotFoundException error)
//         {
//             Console.WriteLine("\n{0}", error.Message);
//             return -1;
//         }
//     }
//     else if (calDatum[calIndex].modulationType == "WLAN")
//     {
//         try
//         {
//             string aiqFile = calDatum[calIndex].modulationFile;
//             string iqsFile = waveDirectory + "Waveforms\\" +
Path.GetFileNameWithoutExtension(aiqFile) + ".iqs";
```

```
//              // Check if .iqs file exists for corresponding .aiq file
//              status = CheckWaveSettingsFile(iqsFile);

//              //////// Need to put an error here if file doesn'e exist ////////

//              // Get waveform data
//              Waveform WlanWaveform = new Waveform();
//              WlanWaveformInfo WlanInfo = WlanWaveform.GetWlanWaveformInfo(iqsFile);

//              // Check whether specified waveform marker is valid
//              if (calDatum[calIndex].WaveConfig.Marker >= 1 &&
calDatum[calIndex].WaveConfig.Marker <= 4) // Marker 1 - 4
//              {
//                  if ((calDatum[calIndex].WaveConfig.Marker == 1 &&
WlanInfo.Marker.Marker1Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 2 &&
WlanInfo.Marker.Marker2Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 3 &&
WlanInfo.Marker.Marker3Info.Count == 0) || (calDatum[calIndex].WaveConfig.Marker == 4 &&
WlanInfo.Marker.Marker4Info.Count == 0))
//                  {
//                      Console.WriteLine("ERROR: Specified waveform marker {0} is undefined for
waveform {1}.", calDatum[calIndex].WaveConfig.Marker, aiqFile);
//                  }
//              }
//              else if (calDatum[calIndex].WaveConfig.Marker == -1) // No marker specified; default to
1st available marker
//              {
//                  if (WlanInfo.Marker.Marker1Info.Count != 0) calDatum[calIndex].WaveConfig.Marker
= 1;
//                  else if (WlanInfo.Marker.Marker2Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 2;
//                  else if (WlanInfo.Marker.Marker3Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 3;
//                  else if (WlanInfo.Marker.Marker4Info.Count != 0)
calDatum[calIndex].WaveConfig.Marker = 4;
//                  else // Error
//                  {
//                      Console.WriteLine("ERROR: No waveform marker(s) defined for waveform {0}.",
aiqFile);
//                  }
//              }
//              else if (calDatum[calIndex].WaveConfig.Marker != 0) // Marker 0 = external marker
```

```
//              {
//                  Console.WriteLine("ERROR: Invalid waveform marker {0} for waveform {1}.",
calDatum[calIndex].WaveConfig.Marker, aiqFile);
//              }

//              double bandwidth = 0;
//              if (WlanInfo.Bandwidth == ChannelBandwidth.FiveMHz)
//              {
//                  bandwidth = 5e6;
//                  calDatum[calIndex].WaveConfig.Bandwidth = ChannelBandwidth.FiveMHz;
//              }
//              else if (WlanInfo.Bandwidth == ChannelBandwidth.TenMHz)
//              {
//                  bandwidth = 10e6;
//                  calDatum[calIndex].WaveConfig.Bandwidth = ChannelBandwidth.TenMHz;
//              }
//              else if (WlanInfo.Bandwidth == ChannelBandwidth.TwentyMHz)
//              {
//                  bandwidth = 20e6;
//                  calDatum[calIndex].WaveConfig.Bandwidth = ChannelBandwidth.TwentyMHz;
//              }
//              else if (WlanInfo.Bandwidth == ChannelBandwidth.FortyMHz)
//              {
//                  bandwidth = 40e6;
//                  calDatum[calIndex].WaveConfig.Bandwidth = ChannelBandwidth.FortyMHz;
//              }
//              else if (WlanInfo.Bandwidth == ChannelBandwidth.EightyMHz)
//              {
//                  bandwidth = 80e6;
//                  calDatum[calIndex].WaveConfig.Bandwidth = ChannelBandwidth.EightyMHz;
//              }
//              else if (WlanInfo.Bandwidth == ChannelBandwidth.OneHundredSixtyMHz)
//              {
//                  bandwidth = 160e6;
//                  calDatum[calIndex].WaveConfig.Bandwidth =
ChannelBandwidth.OneHundredSixtyMHz;
//              }
//              else
//              {
//                  // Error
//              }
```

```
//              double chip = 1 / bandwidth;
//              double burstTime = WlanInfo.Marker.Marker1Info[0].OffTime * chip;
//              double idleTime = WlanInfo.IdleTime;
//              double dutyCycle = (burstTime / (burstTime + idleTime)) * 100;

//              calDatum[calIndex].WaveConfig.Modulation = ModulationType.WLAN;
//              calDatum[calIndex].WaveConfig.WaveformFile = aiqFile;
//              calDatum[calIndex].WaveConfig.Marker = 1; //ushort.Parse(line.Split(',')[1]);
//              calDatum[calIndex].WaveConfig.MarkerLocation = 0;
//              calDatum[calIndex].WaveConfig.DataLocation = 0;
//              calDatum[calIndex].WaveConfig.DataLength = burstTime;
//              calDatum[calIndex].WaveConfig.DutyCycle = dutyCycle;
//              calDatum[calIndex].WaveConfig.MeasOffset = 0; //calDatum[calIndex].DataLocation -
calDatum[calIndex].MarkerLocation;
//              calDatum[calIndex].WaveConfig.MeasLength = burstTime;
//calDatum[calIndex].MeasOffset + calDatum[calIndex].DataLength;
//              calDatum[calIndex].WaveConfig.FrameLength = burstTime + idleTime;
//calDatum[calIndex].MeasLength + double.Parse(line.Split(',')[5]);
//              calDatum[calIndex].WaveConfig.MinSampleFreq = 2 * bandwidth;
//              calDatum[calIndex].WaveConfig.RecSampleFreq = 2 * bandwidth;
//              calDatum[calIndex].WaveConfig.MeasSpan = 0;
//              calDatum[calIndex].WaveConfig.Headroom = 13;
//          }
//          catch (FileNotFoundException error)
//          {
//              Console.WriteLine("\n{0}", error.Message);
//              return -1;
//          }
//      }
//      else if (calDatum[calIndex].modulationType == "NR") // 5G
//      {
//          try
//          {
//              string rfwsName = calDatum[calIndex].modulationFile;
//              string rfwsDirectory = waveDirectory + "Waveforms\\" +
Path.GetFileNameWithoutExtension(rfwsName) + "\\";
//              string rfwsFile = rfwsDirectory + rfwsName + ".rfws";

//              // Check if .rfws file exists
//              status = CheckWaveSettingsFile(rfwsFile);

//              //////// Need to put an error here if file doesn'e exist ////////
```

```
//              // Get waveform data
//              Waveform NrWaveform = new Waveform();
//              NrWaveformInfo NrInfo = NrWaveform.GetNrWaveformInfo(rfwsFile);

//              int maxSlots = NrInfo.SlotEnabled.Length;
//              const double frameLength = 10e-3;
//              double slotLength = 10e-3 / maxSlots;

//              double[] slotLocation = new double[maxSlots];
//              for (int slotIndex = 0; slotIndex < maxSlots; slotIndex++)
//              {
//                  slotLocation[slotIndex] = Math.Round(slotIndex * slotLength, 12);
//              }

//              calDatum[calIndex].WaveConfig.Marker = 1;

//              // Check whether specified waveform marker is valid
//              //if (calDatum[calIndex].WaveConfig.Marker < 1 ||
calDatum[calIndex].WaveConfig.Marker > 4)
//              //{
//              //    // Issue warnign, default to Marker 1
//              //    // Should do additional error checking to see if Marker 1 is defined
//              //    calDatum[calIndex].WaveConfig.Marker = -1;
//              //}

//              // Check whether to use specific slot for capture/analysis
//              bool analyzeSpecificSlot = false;
//              if (calDatum[calIndex].WaveConfig.AnalysisSlot != -1)
//              {
//                  if (calDatum[calIndex].WaveConfig.AnalysisSlot >= 1 &&
calDatum[calIndex].WaveConfig.AnalysisSlot <= maxSlots)
//                  {
//                      analyzeSpecificSlot = true;
//                  }
//                  else
//                  {
//                      // Error
//                      // Requested analysis slot out of range (1 - 8)

//                  }
//              }
```

```
//              // Determine how many slots are enabled
//              int numOfSlotsEnabled = 0;
//              bool specificSlotEnabled = false;
//              for (int slot = 0; slot < maxSlots; slot++)
//              {
//                  if (NrInfo.SlotEnabled[slot] == true)
//                  {
//                      numOfSlotsEnabled++;
//                      if (slot + 1 == calDatum[calIndex].WaveConfig.AnalysisSlot) specificSlotEnabled =
true;
//                  }
//              }

//              if (numOfSlotsEnabled <= 0)
//              {
//                  // Error
//                  // No slots enabled
//              }

//              if (analyzeSpecificSlot && !specificSlotEnabled)
//              {
//                  // Error
//                  // Requested analysis slot is not enabled
//              }

//              // Generate list of enabled slots
//              int index = 0;
//              int[] enabledSlots = new int[numOfSlotsEnabled];
//              for (int slotIndex = 0; slotIndex < maxSlots; slotIndex++)
//              {
//                  if (NrInfo.SlotEnabled[slotIndex] == true)
//                  {
//                      enabledSlots[index++] = slotIndex + 1;
//                  }
//              }

//              int numOfTriggers = 0;
//              double triggerLocation = 0;
//              if (calDatum[calIndex].WaveConfig.Marker == 1) numOfTriggers =
NrInfo.Marker.Marker1Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 2) numOfTriggers =
```

```
NrInfo.Marker.Marker2Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 3) numOfTriggers =
NrInfo.Marker.Marker3Info.Count;
//              else if (calDatum[calIndex].WaveConfig.Marker == 4) numOfTriggers =
NrInfo.Marker.Marker4Info.Count;

//              double[] offsets = new double[numOfTriggers];

//              if (numOfTriggers == 0)
//              {
//                  // Error
//              }
//              else if (numOfTriggers == 1)
//              {
//                  if (calDatum[calIndex].WaveConfig.Marker == 1) triggerLocation =
NrInfo.Marker.Marker1Info[0].OnTime;
//                  else if (calDatum[calIndex].WaveConfig.Marker == 2) triggerLocation =
NrInfo.Marker.Marker2Info[0].OnTime;
//                  else if (calDatum[calIndex].WaveConfig.Marker == 3) triggerLocation =
NrInfo.Marker.Marker3Info[0].OnTime;
//                  else if (calDatum[calIndex].WaveConfig.Marker == 4) triggerLocation =
NrInfo.Marker.Marker4Info[0].OnTime;

//                  if (analyzeSpecificSlot)
//                  {
//                      offsets[0] = slotLocation[calDatum[calIndex].WaveConfig.AnalysisSlot - 1] -
triggerLocation;
//                      if (offsets[0] < 0) offsets[0] += frameLength; // TODO: Need to check if negative
offsets are valid
//                  }
//                  else
//                  {
//                      // Trigger is located at the start of each enabled slot
//                      calDatum[calIndex].WaveConfig.MeasOffset = 0;

//                      //bool slotFound = false;

//                      //// Find offset from trigger location to first enabled slot
//                      //for (int slotIndex = 0; slotIndex < enabledSlots.Length; slotIndex++)
//                      //{
//                      //    if (slotLocation[enabledSlots[slotIndex] - 1] >= triggerLocation)
//                      //    {
```

```
//                  //      offsets[0] = slotLocation[enabledSlots[slotIndex] - 1] - triggerLocation;
//                  //      slotFound = true;
//                  //  }

//                  //  if (slotFound) break;
//                  //}

//                  //// Find offset from trigger location to first enabled slot preceeding the trigger
(uncommon)
//                  //if (!slotFound)
//                  //{
//                  //  offsets[0] = (slotLocation[enabledSlots[0] - 1] - triggerLocation) + frameLength;
//                  //  slotFound = true;
//                  //}
//              }

//                  calDatum[calIndex].WaveConfig.MeasOffset = offsets[0];
//              }
//              else if (numOfTriggers == numOfSlotsEnabled) // Assumes each trigger marks the
location of an enabled slot
//              {
//                  // Trigger is located at the start of each enabled slot
//                  calDatum[calIndex].WaveConfig.MeasOffset = 0;

//                  //// Calculate trigger offsets
//                  //for (int slotIndex = 0; slotIndex < enabledSlots.Length; slotIndex++)
//                  //{
//                  //  if (calDatum[calIndex].WaveConfig.Marker == 1) offsets[slotIndex] =
slotLocation[enabledSlots[slotIndex] - 1] - (NrInfo.Marker.Marker1Info[slotIndex].OnTime *
slotLength);
//                  //  else if (calDatum[calIndex].WaveConfig.Marker == 2) offsets[slotIndex] =
slotLocation[enabledSlots[slotIndex] - 1] - (NrInfo.Marker.Marker2Info[slotIndex].OnTime *
slotLength);
//                  //  else if (calDatum[calIndex].WaveConfig.Marker == 3) offsets[slotIndex] =
slotLocation[enabledSlots[slotIndex] - 1] - (NrInfo.Marker.Marker3Info[slotIndex].OnTime *
slotLength);
//                  //  else if (calDatum[calIndex].WaveConfig.Marker == 4) offsets[slotIndex] =
slotLocation[enabledSlots[slotIndex] - 1] - (NrInfo.Marker.Marker4Info[slotIndex].OnTime *
slotLength);
//                  //}

//                  //// Remove unnecessary digits from small double-precision value
```

```
//                  //double[] tmpOffsets = new double[offsets.Length];
//                  //for (int i = 0; i < offsets.Length; i++)
//                  //{
//                  //    tmpOffsets[i] = Math.Round(offsets[i], 12); // Round to picoseconds
//                  //}

//                  //// Check if trigger offsets are consistent
//                  //bool offsetMismatch = false;
//                  //if (tmpOffsets.Length > 1)
//                  //{
//                  //    for (int i = 1; i < tmpOffsets.Length; i++)
//                  //    {
//                  //        if (tmpOffsets[i] != tmpOffsets[i - 1]) offsetMismatch = true;
//                  //    }
//                  //}

//                  //// All trigger offsets are uniform, so set MeasOffset to any offset value
//                  //if (!offsetMismatch)
//                  //{
//                  //    calDatum[calIndex].WaveConfig.MeasOffset = tmpOffsets[0];
//                  //}
//                  //else
//                  //{
//                  //    // Error
//                  //}
//              }
//              //else if (numOfTriggers < numOfSlotsEnabled)
//              //{
//                  //    double[,] offsetsAll = new double[numOfTriggers, numOfSlotsEnabled];

//                  //    // Calculate all offsets from trigger location to each enabled slot
//                  //    for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//                  //    {
//                  //        for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//                  //        {
//                  //            if (calDatum[calIndex].WaveConfig.Marker == 1) offsetsAll[trigIndex, slotIndex]
= slotLocation[enabledSlots[slotIndex] - 1] - NrInfo.Marker.Marker1Info[trigIndex].OnTime;
//                  //            else if (calDatum[calIndex].WaveConfig.Marker == 2) offsetsAll[trigIndex,
slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] -
NrInfo.Marker.Marker2Info[trigIndex].OnTime;
//                  //            else if (calDatum[calIndex].WaveConfig.Marker == 3) offsetsAll[trigIndex,
slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] -
```

```
NrInfo.Marker.Marker3Info[trigIndex].OnTime;
//              //          else if (calDatum[calIndex].WaveConfig.Marker == 4) offsetsAll[trigIndex,
slotIndex] = slotLocation[enabledSlots[slotIndex] - 1] -
NrInfo.Marker.Marker4Info[trigIndex].OnTime;

//              //          offsetsAll[trigIndex, slotIndex] = Math.Round(offsetsAll[trigIndex, slotIndex], 9);
// Round to picoseconds
//              //      }
//              //  }

//              //  // Find first valid (non-negative) offset for each trigger pulse
//              //  int[] offsetPosition = new int[numOfTriggers];
//              //  for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              //  {
//              //      for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//              //      {
//              //          if (offsetsAll[trigIndex, slotIndex] >= 0)
//              //          {
//              //              offsetPosition[trigIndex] = slotIndex;
//              //              break;
//              //          }
//              //      }
//              //  }

//              //  // Arrange/align offsets starting with valid (non-negative) values
//              //  double[,] tmpOffsetsAllAligned = new double[numOfTriggers, numOfSlotsEnabled];
//              //  for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              //  {
//              //      int slotPosition = offsetPosition[trigIndex];

//              //      for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//              //      {
//              //          if (slotPosition >= numOfSlotsEnabled) slotPosition = 0;
//              //          tmpOffsetsAllAligned[trigIndex, slotIndex] = offsetsAll[trigIndex, slotPosition];
//              //          slotPosition++;
//              //      }
//              //  }

//              //  // Convert negative offsets to positive values
//              //  for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              //  {
//              //      for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
```

```
//              //      {
//              //          if (tmpOffsetsAllAligned[trigIndex, slotIndex] < 0)
tmpOffsetsAllAligned[trigIndex, slotIndex] += frameLength;
//              //      }
//              //  }

//              //  // Remove unnecessary digits from small double-precision value
//              //  for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              //  {
//              //      for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//              //      {
//              //          tmpOffsetsAllAligned[trigIndex, slotIndex] =
Math.Round(tmpOffsetsAllAligned[trigIndex, slotIndex], 9); // Round to picoseconds
//              //      }
//              //  }

//              //  // Store trigger offsets for enabled slots within trigger(n) and trigger(n+1)
//              //  double trigPeriod = 0;
//              //  double trigPeriodMax = frameLength;
//              //  for (int trigIndex = 1; trigIndex <= numOfTriggers; trigIndex++)
//              //  {
//              //      if (trigIndex < numOfTriggers) // Calculate amount of time between adjacent
triggers
//              //      {
//              //          if (calDatum[calIndex].WaveConfig.Marker == 1) trigPeriod =
NrInfo.Marker.Marker1Info[trigIndex].OnTime - NrInfo.Marker.Marker1Info[trigIndex - 1].OnTime;
//              //          else if (calDatum[calIndex].WaveConfig.Marker == 2) trigPeriod =
NrInfo.Marker.Marker2Info[trigIndex].OnTime - NrInfo.Marker.Marker2Info[trigIndex - 1].OnTime;
//              //          else if (calDatum[calIndex].WaveConfig.Marker == 3) trigPeriod =
NrInfo.Marker.Marker3Info[trigIndex].OnTime - NrInfo.Marker.Marker3Info[trigIndex - 1].OnTime;
//              //          else if (calDatum[calIndex].WaveConfig.Marker == 4) trigPeriod =
NrInfo.Marker.Marker4Info[trigIndex].OnTime - NrInfo.Marker.Marker4Info[trigIndex - 1].OnTime;
//              //      }
//              //      else if (trigIndex == numOfTriggers) // Calculate amount of time between last
trigger and first trigger (waveform wrap-around)
//              //      {
//              //          if (calDatum[calIndex].WaveConfig.Marker == 1) trigPeriod = frameLength -
NrInfo.Marker.Marker1Info[trigIndex - 1].OnTime + NrInfo.Marker.Marker1Info[0].OnTime;
//              //          else if (calDatum[calIndex].WaveConfig.Marker == 2) trigPeriod =
frameLength - NrInfo.Marker.Marker2Info[trigIndex - 1].OnTime +
NrInfo.Marker.Marker2Info[0].OnTime;
//              //          else if (calDatum[calIndex].WaveConfig.Marker == 3) trigPeriod =
```

```
frameLength - NrInfo.Marker.Marker3Info[trigIndex - 1].OnTime +
NrInfo.Marker.Marker3Info[0].OnTime;
//              //          else if (calDatum[calIndex].WaveConfig.Marker == 4) trigPeriod =
frameLength - NrInfo.Marker.Marker4Info[trigIndex - 1].OnTime +
NrInfo.Marker.Marker4Info[0].OnTime;
//              //      }

//              //      trigPeriod = Math.Round(trigPeriod, 9); // Round to picoseconds
//              //      if (trigPeriod < trigPeriodMax) trigPeriodMax = trigPeriod;
//              //  }

//              //  // Ignore offsets that are beyond the next consecutive trigger
//              //  for (int trigIndex = 0; trigIndex < numOfTriggers; trigIndex++)
//              //  {
//              //      for (int slotIndex = 0; slotIndex < numOfSlotsEnabled; slotIndex++)
//              //      {
//              //          if (tmpOffsetsAllAligned[trigIndex, slotIndex] >= trigPeriodMax)
tmpOffsetsAllAligned[trigIndex, slotIndex] = -9999;
//              //      }
//              //  }

//              //  // Check if trigger offsets are consistent
//              //  bool offsetMismatch = false;
//              //  if (tmpOffsetsAllAligned.GetLength(0) > 1)
//              //  {
//              //      for (int trigIndex = 1; trigIndex < tmpOffsetsAllAligned.GetLength(0); trigIndex++)
//              //      {
//              //          for (int slotIndex = 0; slotIndex < tmpOffsetsAllAligned.GetLength(1);
slotIndex++)
//              //          {
//              //              if (tmpOffsetsAllAligned[trigIndex, slotIndex] != -9999 &&
tmpOffsetsAllAligned[trigIndex, slotIndex] != tmpOffsetsAllAligned[trigIndex - 1, slotIndex])
offsetMismatch = true;
//              //          }
//              //      }
//              //  }

//              //  // All trigger offsets are the same, so set MeasOffset to first enabled slot
//              //  if (!offsetMismatch)
//              //  {
//              //      calDatum[calIndex].WaveConfig.MeasOffset = tmpOffsetsAllAligned[0, 0];
//              //  }
```

```
//              //    else
//              //    {
//              //        // Error
//              //    }
//              //}
//              else if (numOfTriggers > numOfSlotsEnabled)
//              {
//                  // ???
//              }

//              calDatum[calIndex].WaveConfig.Modulation = ModulationType.NR;
//              calDatum[calIndex].WaveConfig.WaveformFile = rfwsFile;
//              calDatum[calIndex].WaveConfig.Bandwidth = NrInfo.Bandwidth;
//              calDatum[calIndex].WaveConfig.FrameLength = frameLength; // 10e-3;
//              calDatum[calIndex].WaveConfig.DataLength = slotLength;
//              calDatum[calIndex].WaveConfig.MeasLength =
calDatum[calIndex].WaveConfig.MeasOffset + calDatum[calIndex].WaveConfig.DataLength;
//              calDatum[calIndex].WaveConfig.DutyCycle = (slotLength * numOfSlotsEnabled /
frameLength) * 100;

//              //GetModulationSamplingParams(calDatum[calIndex].WaveConfig.Modulation,
calDatum[calIndex].srcFreq, calDatum[calIndex].WaveConfig.Bandwidth, calIndex, out double
minSampleFreq, out double recSampleFreq, out double measSpan);

//              double sampleFreq = 0;
//              if (NrInfo.Bandwidth == ChannelBandwidth.TenMHz) sampleFreq = 20e6;
//              else if (NrInfo.Bandwidth == ChannelBandwidth.FifteenMHz) sampleFreq = 30e6;
//              else if (NrInfo.Bandwidth == ChannelBandwidth.TwentyMHz) sampleFreq = 40e6;
//              else if (NrInfo.Bandwidth == ChannelBandwidth.FortyMHz) sampleFreq = 80e6;
//              else if (NrInfo.Bandwidth == ChannelBandwidth.FiftyMHz) sampleFreq = 100e6;
//              else if (NrInfo.Bandwidth == ChannelBandwidth.SixtyMHz) sampleFreq = 120e6;
//              else if (NrInfo.Bandwidth == ChannelBandwidth.EightyMHz) sampleFreq = 160e6;
//              else if (NrInfo.Bandwidth == ChannelBandwidth.OneHundredMHz) sampleFreq =
200e6;

//              calDatum[calIndex].WaveConfig.MinSampleFreq = sampleFreq; //minSampleFreq;
//              calDatum[calIndex].WaveConfig.RecSampleFreq = sampleFreq; //recSampleFreq;
//              calDatum[calIndex].WaveConfig.MeasSpan = 0; //measSpan;
//              calDatum[calIndex].WaveConfig.Headroom = 10;
//          }
//          catch (FileNotFoundException error)
//          {
```

```
//                Console.WriteLine("\n{0}", error.Message);
//                return -1;
//            }
//        }
//    }

//    return 0;
//}

//private void GetModulationSamplingParams(ModulationType modType, double srcFreq,
ChannelBandwidth bandwidth, int index, out double minSampleFreq, out double recSampleFreq,
out double measSpan)
//{
//    LteMeasurement tests = LteMeasurement.MeasAclr | LteMeasurement.MeasModAccuracy;
//    minSampleFreq = double.NaN;
//    recSampleFreq = double.NaN;
//    measSpan = double.NaN;

//    if (modType == ModulationType.LTE_FDD)
//    {
//        LteFddAnalysis lteFddUtraAnalysis = new LteFddAnalysis();
//        lteFddUtraAnalysis.centerFreq = srcFreq;
//        lteFddUtraAnalysis.sampleFreq = 100e6;
//        lteFddUtraAnalysis.numOfSamples = 17050;
//        lteFddUtraAnalysis.mode = AclrMode.Utra;
//        lteFddUtraAnalysis.numOfChans = 5;
//        lteFddUtraAnalysis.numOfSlots = 1;
//        if (bandwidth == ChannelBandwidth.OnePointFourMHz) lteFddUtraAnalysis.eutraBW = 1.4f;
//        else if (bandwidth == ChannelBandwidth.ThreeMHz) lteFddUtraAnalysis.eutraBW = 3;
//        else if (bandwidth == ChannelBandwidth.FiveMHz) lteFddUtraAnalysis.eutraBW = 5;
//        else if (bandwidth == ChannelBandwidth.TenMHz) lteFddUtraAnalysis.eutraBW = 10;
//        else if (bandwidth == ChannelBandwidth.FifteenMHz) lteFddUtraAnalysis.eutraBW = 15;
//        else if (bandwidth == ChannelBandwidth.TwentyMHz) lteFddUtraAnalysis.eutraBW = 20;
//        lteFddUtraAnalysis.AnalysisSetup();
//        lteFddUtraAnalysis.getConstraints(tests, out double minSampleFreqUtra, out double
recSampleFreqUtra, out double minMeasSpanUtra);

//        LteFddAnalysis lteFddEutraAnalysis = new LteFddAnalysis();
//        lteFddEutraAnalysis.centerFreq = srcFreq;
//        lteFddEutraAnalysis.sampleFreq = 100e6;
//        lteFddEutraAnalysis.numOfSamples = 17050;
//        lteFddEutraAnalysis.mode = AclrMode.Eutra;
```

```
//      lteFddEutraAnalysis.numOfChans = 3;
//      lteFddEutraAnalysis.numOfSlots = 1;
//      if (bandwidth == ChannelBandwidth.OnePointFourMHz) lteFddEutraAnalysis.eutraBW =
1.4f;
//      else if (bandwidth == ChannelBandwidth.ThreeMHz) lteFddEutraAnalysis.eutraBW = 3;
//      else if (bandwidth == ChannelBandwidth.FiveMHz) lteFddEutraAnalysis.eutraBW = 5;
//      else if (bandwidth == ChannelBandwidth.TenMHz) lteFddEutraAnalysis.eutraBW = 10;
//      else if (bandwidth == ChannelBandwidth.FifteenMHz) lteFddEutraAnalysis.eutraBW = 15;
//      else if (bandwidth == ChannelBandwidth.TwentyMHz) lteFddEutraAnalysis.eutraBW = 20;
//      lteFddEutraAnalysis.AnalysisSetup();
//      lteFddEutraAnalysis.getConstraints(tests, out double minSampleFreqEutra, out double
recSampleFreqEutra, out double minMeasSpanEutra);

//      minSampleFreq = minSampleFreqUtra > minSampleFreqEutra ? minSampleFreqUtra :
minSampleFreqEutra;
//      recSampleFreq = recSampleFreqUtra > recSampleFreqEutra ? recSampleFreqUtra :
recSampleFreqEutra;
//      measSpan = minMeasSpanUtra > minMeasSpanEutra ? minMeasSpanUtra :
minMeasSpanEutra;
//      if (measSpan % 1 != 0) measSpan = (long)(measSpan + 1); // Round up to avoid long-
precision values
//  }
//  else if (modType == ModulationType.LTE_TDD)
//  {
//      LteTddAnalysis lteTddUtraAnalysis = new LteTddAnalysis();
//      lteTddUtraAnalysis.centerFreq = srcFreq;
//      lteTddUtraAnalysis.sampleFreq = 100e6;
//      lteTddUtraAnalysis.numOfSamples = 17050;
//      lteTddUtraAnalysis.mode = AclrMode.Utra;
//      lteTddUtraAnalysis.numOfChans = 5;
//      lteTddUtraAnalysis.numOfSlots = 1;
//      if (bandwidth == ChannelBandwidth.OnePointFourMHz) lteTddUtraAnalysis.eutraBW = 1.4f;
//      else if (bandwidth == ChannelBandwidth.ThreeMHz) lteTddUtraAnalysis.eutraBW = 3;
//      else if (bandwidth == ChannelBandwidth.FiveMHz) lteTddUtraAnalysis.eutraBW = 5;
//      else if (bandwidth == ChannelBandwidth.TenMHz) lteTddUtraAnalysis.eutraBW = 10;
//      else if (bandwidth == ChannelBandwidth.FifteenMHz) lteTddUtraAnalysis.eutraBW = 15;
//      else if (bandwidth == ChannelBandwidth.TwentyMHz) lteTddUtraAnalysis.eutraBW = 20;
//      lteTddUtraAnalysis.AnalysisSetup();
//      lteTddUtraAnalysis.getConstraints(tests, out double minSampleFreqUtra, out double
recSampleFreqUtra, out double minMeasSpanUtra);

//      LteTddAnalysis lteTddEutraAnalysis = new LteTddAnalysis();
```

```
//      lteTddEutraAnalysis.centerFreq = srcFreq;
//      lteTddEutraAnalysis.sampleFreq = 100e6;
//      lteTddEutraAnalysis.numOfSamples = 17050;
//      lteTddEutraAnalysis.mode = AclrMode.Eutra;
//      lteTddEutraAnalysis.numOfChans = 3;
//      lteTddEutraAnalysis.numOfSlots = 1;
//      if (bandwidth == ChannelBandwidth.OnePointFourMHz) lteTddEutraAnalysis.eutraBW =
1.4f;
//      else if (bandwidth == ChannelBandwidth.ThreeMHz) lteTddEutraAnalysis.eutraBW = 3;
//      else if (bandwidth == ChannelBandwidth.FiveMHz) lteTddEutraAnalysis.eutraBW = 5;
//      else if (bandwidth == ChannelBandwidth.TenMHz) lteTddEutraAnalysis.eutraBW = 10;
//      else if (bandwidth == ChannelBandwidth.FifteenMHz) lteTddEutraAnalysis.eutraBW = 15;
//      else if (bandwidth == ChannelBandwidth.TwentyMHz) lteTddEutraAnalysis.eutraBW = 20;
//      lteTddEutraAnalysis.AnalysisSetup();
//      lteTddEutraAnalysis.getConstraints(tests, out double minSampleFreqEutra, out double
recSampleFreqEutra, out double minMeasSpanEutra);

//      minSampleFreq = minSampleFreqUtra > minSampleFreqEutra ? minSampleFreqUtra :
minSampleFreqEutra;
//      recSampleFreq = recSampleFreqUtra > recSampleFreqEutra ? recSampleFreqUtra :
recSampleFreqEutra;
//      measSpan = minMeasSpanUtra > minMeasSpanEutra ? minMeasSpanUtra :
minMeasSpanEutra;
//      if (measSpan % 1 != 0) measSpan = (long)(measSpan + 1); // Round up to avoid long-
precision values
//  }
//  else if (modType == ModulationType.TDSCDMA)
//  {
//      TdscdmaAnalysis tdscdmaAnalysis = new TdscdmaAnalysis();
//      tdscdmaAnalysis.centerFreq = srcFreq;
//      tdscdmaAnalysis.sampleFreq = 9.68e6;
//      tdscdmaAnalysis.numOfSamples = 55660;
//      tdscdmaAnalysis.AnalysisSetup();
//      tdscdmaAnalysis.getConstraints(TdscdmaMeasurement.MeasModAccuracy, out double
minSampleFreqEVM);
//      tdscdmaAnalysis.getConstraints(TdscdmaMeasurement.MeasAclr, out double
minSampleFreqACLR);

//      minSampleFreq = recSampleFreq = minSampleFreqEVM > minSampleFreqACLR ?
minSampleFreqEVM : minSampleFreqACLR;
//      //measSpan = 0;
//  }
```

```
//}

public double[,] ImportNoiseConfig(string noiseConfigFile)
{
string line = "";
string[] splitLine = null;
double[,] data = new double[2, 20];

try
{
using (StreamReader stream = new StreamReader(noiseConfigFile))
{
Regex CSVParser = new Regex(",(?=(?:[^\"]*\"[^\"]*\")*(?![^\"]*\"))");

// Header information is on rows 1 through 6 of the Noise_Config.csv file
for (int row = 0; row < 6; row++)
{
line = stream.ReadLine(); // Read rows 1 - 6
}

do
{
// ENR data starts on row 7 of the Noise_Config.csv file
for (int freqIndex = 0; freqIndex < 20; freqIndex++)
{
line = stream.ReadLine(); // Read row 7
splitLine = CSVParser.Split(line);
data[0, freqIndex] = double.Parse(splitLine[0]); // Frequency
data[1, freqIndex] = double.Parse(splitLine[1]); // ENR
}

} while ((line = stream.ReadLine()) != null);
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
}

return data;
}
```

```
public List<NoiseData> ImportNoiseData(string noiseDataFile)
{
string line = "";
string[] splitLine = null;
List<NoiseData> caldatum = new List<NoiseData>();

try
{
using (StreamReader stream = new StreamReader(noiseDataFile))
{
Regex CSVParser = new Regex(",(?=(?:[^\"]*\"[^\"]*\")*(?![^\"]*\"))");

// Header information is on rows 1 through 8 of the Cal_Config.csv file
for (int row = 0; row < 8; row++)
{
line = stream.ReadLine(); // Read rows 1 - 8
}

line = stream.ReadLine();

do
{
splitLine = CSVParser.Split(line);

if (splitLine.Length > 1)
{
NoiseData data = new NoiseData();
data.srcSelect = splitLine[0];
data.srcPort = splitLine[1];
data.measPort = splitLine[2];
data.freq = double.Parse(splitLine[3]);
data.ENR = double.Parse(splitLine[4]);
data.Loss = double.Parse(splitLine[5]);
data.On = double.Parse(splitLine[6]);
data.Off = double.Parse(splitLine[7]);

caldatum.Add(data);
}
} while ((line = stream.ReadLine()) != null);
}
}
catch (FileNotFoundException error)
```

```
{
Console.WriteLine("\n{0}", error.Message);
}

return caldatum;
}

public List<NoiseData> ImportNoiseDataVer2(string noiseDataFile)
{
string line = "";
List<NoiseData> caldatum = new List<NoiseData>();

try
{
using (StreamReader stream = new StreamReader(noiseDataFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "NOISE")
{
NoiseData data = new NoiseData();
data.srcSelect = line.Split(',')[0];
data.srcPortMatrix = line.Split(',')[1];
data.srcPortRf = line.Split(',')[2];
data.measPortMatrix = line.Split(',')[3];
data.measPortRf = line.Split(',')[4];
data.freq = double.Parse(line.Split(',')[5]);
data.ENR = double.Parse(line.Split(',')[6]);
data.Loss = double.Parse(line.Split(',')[7]);
data.On = double.Parse(line.Split(',')[8]);
data.Off = double.Parse(line.Split(',')[9]);

caldatum.Add(data);
}
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
```

```
}

return caldatum;
}

public Tuple<List<CalDataRF300>, Info, CalSettings, CoupledPort, SwitchSelect,
ExternalAttenuation> ImportCalConfigRF300(string calConfigFile)
{
string line = "";
Info info = new Info();
CalSettings CalSelect = new CalSettings();
CoupledPort coupledPort = new CoupledPort();
SwitchSelect switchSelect = new SwitchSelect();
ExternalAttenuation attenExternal = new ExternalAttenuation();
List<CalDataRF300> caldatum = new List<CalDataRF300>();

try
{
using (StreamReader stream = new StreamReader(calConfigFile))
{
while ((line = stream.ReadLine()) != null)
{
string tmpString = line.Split(',')[0];

if (tmpString.ToUpper() == "PRODUCT:")
{
info.Product = line.Split(',')[1];
}
if (tmpString.ToUpper() == "REVISION:")
{
info.Revision = line.Split(',')[1];
}
if (tmpString.ToUpper() == "TEST PROGRAM:")
{
info.Program = line.Split(',')[1];
}
if (tmpString.ToUpper() == "CALIBRATION VERSION:")
{
info.CalVersion = line.Split(',')[1];
}
if (tmpString.ToUpper() == "COMMENT:")
{
```

```
info.Comment = line.Split(',')[1];
}
else if (tmpString.ToUpper() == "USE EXTERNAL POWER METER:")
{
CalSelect.PowerMeterAvailable = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE RF SOURCE/MEASURE:")
{
CalSelect.SourceMeasurePath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE INTERNAL DIRECT PATH:")
{
CalSelect.InternalMatrixPath = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "CALIBRATE NOISE SOURCE:")
{
CalSelect.NoiseSource = line.Split(',')[1].ToUpper() == "Y" ? true : false;
}
else if (tmpString.ToUpper() == "DUT_P1-P4 PORT:")
{
tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") coupledPort.DUT_P1_P4 = MeasPort.P1;
else if (tmpString.ToUpper() == "P2") coupledPort.DUT_P1_P4 = MeasPort.P2;
else if (tmpString.ToUpper() == "P3") coupledPort.DUT_P1_P4 = MeasPort.P3;
else if (tmpString.ToUpper() == "P4") coupledPort.DUT_P1_P4 = MeasPort.P4;
else if (tmpString.ToUpper() == "P5") coupledPort.DUT_P1_P4 = MeasPort.P5;
else if (tmpString.ToUpper() == "P6") coupledPort.DUT_P1_P4 = MeasPort.P6;
else if (tmpString.ToUpper() == "P7") coupledPort.DUT_P1_P4 = MeasPort.P7;
else if (tmpString.ToUpper() == "P8") coupledPort.DUT_P1_P4 = MeasPort.P8;
else if (tmpString.ToUpper() == "P9") coupledPort.DUT_P1_P4 = MeasPort.P9;
else if (tmpString.ToUpper() == "P10") coupledPort.DUT_P1_P4 = MeasPort.P10;
else if (tmpString.ToUpper() == "P11") coupledPort.DUT_P1_P4 = MeasPort.P11;
else if (tmpString.ToUpper() == "P12") coupledPort.DUT_P1_P4 = MeasPort.P12;
else if (tmpString.ToUpper() == "P13") coupledPort.DUT_P1_P4 = MeasPort.P13;
else if (tmpString.ToUpper() == "P14") coupledPort.DUT_P1_P4 = MeasPort.P14;
else if (tmpString.ToUpper() == "P15") coupledPort.DUT_P1_P4 = MeasPort.P15;
else if (tmpString.ToUpper() == "P16") coupledPort.DUT_P1_P4 = MeasPort.P16;
else if (tmpString.ToUpper() == "P1_FWD") coupledPort.DUT_P1_P4 = MeasPort.P1_Fwd;
else if (tmpString.ToUpper() == "P2_FWD") coupledPort.DUT_P1_P4 = MeasPort.P2_Fwd;
else if (tmpString.ToUpper() == "P3_FWD") coupledPort.DUT_P1_P4 = MeasPort.P3_Fwd;
else if (tmpString.ToUpper() == "P4_FWD") coupledPort.DUT_P1_P4 = MeasPort.P4_Fwd;
else if (tmpString.ToUpper() == "P5_FWD") coupledPort.DUT_P1_P4 = MeasPort.P5_Fwd;
```

```
else if (tmpString.ToUpper() == "P6_FWD") coupledPort.DUT_P1_P4 = MeasPort.P6_Fwd;
else if (tmpString.ToUpper() == "P7_FWD") coupledPort.DUT_P1_P4 = MeasPort.P7_Fwd;
else if (tmpString.ToUpper() == "P8_FWD") coupledPort.DUT_P1_P4 = MeasPort.P8_Fwd;
else if (tmpString.ToUpper() == "P1_REV") coupledPort.DUT_P1_P4 = MeasPort.P1_Rev;
else if (tmpString.ToUpper() == "P2_REV") coupledPort.DUT_P1_P4 = MeasPort.P2_Rev;
else if (tmpString.ToUpper() == "P3_REV") coupledPort.DUT_P1_P4 = MeasPort.P3_Rev;
else if (tmpString.ToUpper() == "P4_REV") coupledPort.DUT_P1_P4 = MeasPort.P4_Rev;
else if (tmpString.ToUpper() == "P5_REV") coupledPort.DUT_P1_P4 = MeasPort.P5_Rev;
else if (tmpString.ToUpper() == "P6_REV") coupledPort.DUT_P1_P4 = MeasPort.P6_Rev;
else if (tmpString.ToUpper() == "P7_REV") coupledPort.DUT_P1_P4 = MeasPort.P7_Rev;
else if (tmpString.ToUpper() == "P8_REV") coupledPort.DUT_P1_P4 = MeasPort.P8_Rev;
else if (tmpString.ToUpper() == "CAL") coupledPort.DUT_P1_P4 = MeasPort.CAL;
else if (tmpString.ToUpper() == "M1") coupledPort.DUT_P1_P4 = MeasPort.M1;
else if (tmpString.ToUpper() == "M2") coupledPort.DUT_P1_P4 = MeasPort.M2;
else if (tmpString.ToUpper() == "M3") coupledPort.DUT_P1_P4 = MeasPort.M3;
else if (tmpString.ToUpper() == "M4") coupledPort.DUT_P1_P4 = MeasPort.M4;
else if (tmpString.ToUpper() == "M5") coupledPort.DUT_P1_P4 = MeasPort.M5;
else if (tmpString.ToUpper() == "M6") coupledPort.DUT_P1_P4 = MeasPort.M6;
else if (tmpString.ToUpper() == "M7") coupledPort.DUT_P1_P4 = MeasPort.M7;
else if (tmpString.ToUpper() == "M8") coupledPort.DUT_P1_P4 = MeasPort.M8;
else if (tmpString.ToUpper() == "M9") coupledPort.DUT_P1_P4 = MeasPort.M9;
else if (tmpString.ToUpper() == "M10") coupledPort.DUT_P1_P4 = MeasPort.M10;
else if (tmpString.ToUpper() == "M11") coupledPort.DUT_P1_P4 = MeasPort.M11;
else if (tmpString.ToUpper() == "M12") coupledPort.DUT_P1_P4 = MeasPort.M12;
else if (tmpString.ToUpper() == "M13") coupledPort.DUT_P1_P4 = MeasPort.M13;
else if (tmpString.ToUpper() == "M14") coupledPort.DUT_P1_P4 = MeasPort.M14;
else if (tmpString.ToUpper() == "M15") coupledPort.DUT_P1_P4 = MeasPort.M15;
else if (tmpString.ToUpper() == "M16") coupledPort.DUT_P1_P4 = MeasPort.M16;
else if (tmpString.ToUpper() == "M17") coupledPort.DUT_P1_P4 = MeasPort.M17;
else if (tmpString.ToUpper() == "M18") coupledPort.DUT_P1_P4 = MeasPort.M18;
else if (tmpString.ToUpper() == "M19") coupledPort.DUT_P1_P4 = MeasPort.M19;
else if (tmpString.ToUpper() == "M20") coupledPort.DUT_P1_P4 = MeasPort.M20;
else if (tmpString.ToUpper() == "M21") coupledPort.DUT_P1_P4 = MeasPort.M21;
else if (tmpString.ToUpper() == "M22") coupledPort.DUT_P1_P4 = MeasPort.M22;
else if (tmpString.ToUpper() == "M23") coupledPort.DUT_P1_P4 = MeasPort.M23;
else if (tmpString.ToUpper() == "M24") coupledPort.DUT_P1_P4 = MeasPort.M24;
else if (tmpString.ToUpper() == "M25") coupledPort.DUT_P1_P4 = MeasPort.M25;
else if (tmpString.ToUpper() == "M26") coupledPort.DUT_P1_P4 = MeasPort.M26;
else coupledPort.DUT_P1_P4 = MeasPort.NONE;
}
else if (tmpString.ToUpper() == "DUT_P5-P8 PORT:")
{
```

```
tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") coupledPort.DUT_P5_P8 = MeasPort.P1;
else if (tmpString.ToUpper() == "P2") coupledPort.DUT_P5_P8 = MeasPort.P2;
else if (tmpString.ToUpper() == "P3") coupledPort.DUT_P5_P8 = MeasPort.P3;
else if (tmpString.ToUpper() == "P4") coupledPort.DUT_P5_P8 = MeasPort.P4;
else if (tmpString.ToUpper() == "P5") coupledPort.DUT_P5_P8 = MeasPort.P5;
else if (tmpString.ToUpper() == "P6") coupledPort.DUT_P5_P8 = MeasPort.P6;
else if (tmpString.ToUpper() == "P7") coupledPort.DUT_P5_P8 = MeasPort.P7;
else if (tmpString.ToUpper() == "P8") coupledPort.DUT_P5_P8 = MeasPort.P8;
else if (tmpString.ToUpper() == "P9") coupledPort.DUT_P5_P8 = MeasPort.P9;
else if (tmpString.ToUpper() == "P10") coupledPort.DUT_P5_P8 = MeasPort.P10;
else if (tmpString.ToUpper() == "P11") coupledPort.DUT_P5_P8 = MeasPort.P11;
else if (tmpString.ToUpper() == "P12") coupledPort.DUT_P5_P8 = MeasPort.P12;
else if (tmpString.ToUpper() == "P13") coupledPort.DUT_P5_P8 = MeasPort.P13;
else if (tmpString.ToUpper() == "P14") coupledPort.DUT_P5_P8 = MeasPort.P14;
else if (tmpString.ToUpper() == "P15") coupledPort.DUT_P5_P8 = MeasPort.P15;
else if (tmpString.ToUpper() == "P16") coupledPort.DUT_P5_P8 = MeasPort.P16;
else if (tmpString.ToUpper() == "P1_FWD") coupledPort.DUT_P5_P8 = MeasPort.P1_Fwd;
else if (tmpString.ToUpper() == "P2_FWD") coupledPort.DUT_P5_P8 = MeasPort.P2_Fwd;
else if (tmpString.ToUpper() == "P3_FWD") coupledPort.DUT_P5_P8 = MeasPort.P3_Fwd;
else if (tmpString.ToUpper() == "P4_FWD") coupledPort.DUT_P5_P8 = MeasPort.P4_Fwd;
else if (tmpString.ToUpper() == "P5_FWD") coupledPort.DUT_P5_P8 = MeasPort.P5_Fwd;
else if (tmpString.ToUpper() == "P6_FWD") coupledPort.DUT_P5_P8 = MeasPort.P6_Fwd;
else if (tmpString.ToUpper() == "P7_FWD") coupledPort.DUT_P5_P8 = MeasPort.P7_Fwd;
else if (tmpString.ToUpper() == "P8_FWD") coupledPort.DUT_P5_P8 = MeasPort.P8_Fwd;
else if (tmpString.ToUpper() == "P1_REV") coupledPort.DUT_P5_P8 = MeasPort.P1_Rev;
else if (tmpString.ToUpper() == "P2_REV") coupledPort.DUT_P5_P8 = MeasPort.P2_Rev;
else if (tmpString.ToUpper() == "P3_REV") coupledPort.DUT_P5_P8 = MeasPort.P3_Rev;
else if (tmpString.ToUpper() == "P4_REV") coupledPort.DUT_P5_P8 = MeasPort.P4_Rev;
else if (tmpString.ToUpper() == "P5_REV") coupledPort.DUT_P5_P8 = MeasPort.P5_Rev;
else if (tmpString.ToUpper() == "P6_REV") coupledPort.DUT_P5_P8 = MeasPort.P6_Rev;
else if (tmpString.ToUpper() == "P7_REV") coupledPort.DUT_P5_P8 = MeasPort.P7_Rev;
else if (tmpString.ToUpper() == "P8_REV") coupledPort.DUT_P5_P8 = MeasPort.P8_Rev;
else if (tmpString.ToUpper() == "CAL") coupledPort.DUT_P5_P8 = MeasPort.CAL;
else if (tmpString.ToUpper() == "M1") coupledPort.DUT_P5_P8 = MeasPort.M1;
else if (tmpString.ToUpper() == "M2") coupledPort.DUT_P5_P8 = MeasPort.M2;
else if (tmpString.ToUpper() == "M3") coupledPort.DUT_P5_P8 = MeasPort.M3;
else if (tmpString.ToUpper() == "M4") coupledPort.DUT_P5_P8 = MeasPort.M4;
else if (tmpString.ToUpper() == "M5") coupledPort.DUT_P5_P8 = MeasPort.M5;
else if (tmpString.ToUpper() == "M6") coupledPort.DUT_P5_P8 = MeasPort.M6;
else if (tmpString.ToUpper() == "M7") coupledPort.DUT_P5_P8 = MeasPort.M7;
else if (tmpString.ToUpper() == "M8") coupledPort.DUT_P5_P8 = MeasPort.M8;
```

```
else if (tmpString.ToUpper() == "M9") coupledPort.DUT_P5_P8 = MeasPort.M9;
else if (tmpString.ToUpper() == "M10") coupledPort.DUT_P5_P8 = MeasPort.M10;
else if (tmpString.ToUpper() == "M11") coupledPort.DUT_P5_P8 = MeasPort.M11;
else if (tmpString.ToUpper() == "M12") coupledPort.DUT_P5_P8 = MeasPort.M12;
else if (tmpString.ToUpper() == "M13") coupledPort.DUT_P5_P8 = MeasPort.M13;
else if (tmpString.ToUpper() == "M14") coupledPort.DUT_P5_P8 = MeasPort.M14;
else if (tmpString.ToUpper() == "M15") coupledPort.DUT_P5_P8 = MeasPort.M15;
else if (tmpString.ToUpper() == "M16") coupledPort.DUT_P5_P8 = MeasPort.M16;
else if (tmpString.ToUpper() == "M17") coupledPort.DUT_P5_P8 = MeasPort.M17;
else if (tmpString.ToUpper() == "M18") coupledPort.DUT_P5_P8 = MeasPort.M18;
else if (tmpString.ToUpper() == "M19") coupledPort.DUT_P5_P8 = MeasPort.M19;
else if (tmpString.ToUpper() == "M20") coupledPort.DUT_P5_P8 = MeasPort.M20;
else if (tmpString.ToUpper() == "M21") coupledPort.DUT_P5_P8 = MeasPort.M21;
else if (tmpString.ToUpper() == "M22") coupledPort.DUT_P5_P8 = MeasPort.M22;
else if (tmpString.ToUpper() == "M23") coupledPort.DUT_P5_P8 = MeasPort.M23;
else if (tmpString.ToUpper() == "M24") coupledPort.DUT_P5_P8 = MeasPort.M24;
else if (tmpString.ToUpper() == "M25") coupledPort.DUT_P5_P8 = MeasPort.M25;
else if (tmpString.ToUpper() == "M26") coupledPort.DUT_P5_P8 = MeasPort.M26;
else coupledPort.DUT_P5_P8 = MeasPort.NONE;
}
else if (tmpString.ToUpper() == "SWITCH CONFIGURATION:")
{
line = stream.ReadLine();

const int switchCount = 8;

for (int switchIndex = 0; switchIndex < switchCount; switchIndex++)
{
tmpString = line.Split(',')[switchIndex + 1];

if (tmpString.ToUpper() == "BB") switchSelect.switchPosition[switchIndex] =
SwitchSelect.SwitchPosition.BB;
else if (tmpString.ToUpper() == "FILT") switchSelect.switchPosition[switchIndex] =
SwitchSelect.SwitchPosition.FILT;
else if (tmpString.ToUpper() == "HB") switchSelect.switchPosition[switchIndex] =
SwitchSelect.SwitchPosition.HB;
else if (tmpString.ToUpper() == "HB_DESELECT") switchSelect.switchPosition[switchIndex] =
SwitchSelect.SwitchPosition.HB_Deselect;
else if (tmpString.ToUpper() == "LB_HPF") switchSelect.switchPosition[switchIndex] =
SwitchSelect.SwitchPosition.LB_HPF;
else switchSelect.switchPosition[switchIndex] = SwitchSelect.SwitchPosition.None;
}
```

```
}
else if (tmpString.ToUpper() == "PORT MODULE ATTENUATION:")
{
line = stream.ReadLine();

const int srcPathCount = 17;

for (int srcIndex = 0; srcIndex < srcPathCount; srcIndex++)
{
attenExternal.srcAttenPortModule[srcIndex] = double.Parse(line.Split(',')[srcIndex + 1]);
}

const int measPathCount = 26;

for (int measIndex = 0; measIndex < measPathCount; measIndex++)
{
attenExternal.measAttenPortModule[measIndex] = double.Parse(line.Split(',')[srcPathCount +
measIndex + 1]);
}
}
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG2" || tmpString.ToUpper() ==
"NOISE")
{
CalDataRF300 caldata = new CalDataRF300();

// Source
tmpString = line.Split(',')[0];
if (tmpString.ToUpper() == "SG1") caldata.srcSelect = SigGen.SG1;
else if (tmpString.ToUpper() == "SG2") caldata.srcSelect = SigGen.SG2;
else if (tmpString.ToUpper() == "NOISE") caldata.srcSelect = SigGen.NOISE;

// Source Port
tmpString = line.Split(',')[1];
if (tmpString.ToUpper() == "P1") caldata.srcPort = SrcPort.P1;
else if (tmpString.ToUpper() == "P2") caldata.srcPort = SrcPort.P2;
else if (tmpString.ToUpper() == "P3") caldata.srcPort = SrcPort.P3;
else if (tmpString.ToUpper() == "P4") caldata.srcPort = SrcPort.P4;
else if (tmpString.ToUpper() == "P5") caldata.srcPort = SrcPort.P5;
else if (tmpString.ToUpper() == "P6") caldata.srcPort = SrcPort.P6;
else if (tmpString.ToUpper() == "P7") caldata.srcPort = SrcPort.P7;
else if (tmpString.ToUpper() == "P8") caldata.srcPort = SrcPort.P8;
else if (tmpString.ToUpper() == "P9") caldata.srcPort = SrcPort.P9;
```

```
else if (tmpString.ToUpper() == "P10") caldata.srcPort = SrcPort.P10;
else if (tmpString.ToUpper() == "P11") caldata.srcPort = SrcPort.P11;
else if (tmpString.ToUpper() == "P12") caldata.srcPort = SrcPort.P12;
else if (tmpString.ToUpper() == "P13") caldata.srcPort = SrcPort.P13;
else if (tmpString.ToUpper() == "P14") caldata.srcPort = SrcPort.P14;
else if (tmpString.ToUpper() == "P15") caldata.srcPort = SrcPort.P15;
else if (tmpString.ToUpper() == "P16") caldata.srcPort = SrcPort.P16;
else if (tmpString.ToUpper() == "CAL") caldata.srcPort = SrcPort.CAL;
else if (tmpString.ToUpper() == "SG1" || tmpString.ToUpper() == "SG1_OUT") caldata.srcPort =
SrcPort.SG1_OUT;
else caldata.srcPort = SrcPort.NONE;

// Source Port Alias
caldata.srcPortAlias = line.Split(',')[2];

// Amp Port
tmpString = line.Split(',')[3];
if (tmpString.ToUpper() == "A1" || tmpString.ToUpper() == "HPA1") caldata.ampPort =
CalDataRF300.AmpPort.HPA1;
else if (tmpString.ToUpper() == "A2" || tmpString.ToUpper() == "HPA2") caldata.ampPort =
CalDataRF300.AmpPort.HPA2;
else if (tmpString.ToUpper() == "A3" || tmpString.ToUpper() == "HPA3") caldata.ampPort =
CalDataRF300.AmpPort.HPA3;
else if (tmpString.ToUpper() == "A4" || tmpString.ToUpper() == "HPA4") caldata.ampPort =
CalDataRF300.AmpPort.HPA4;
else caldata.ampPort = CalDataRF300.AmpPort.None;

// Source MT-RF300 Primary
tmpString = line.Split(',')[4];
if (tmpString.ToUpper() == "DUT_P1") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P1;
else if (tmpString.ToUpper() == "DUT_P2") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P2;
else if (tmpString.ToUpper() == "DUT_P3") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P3;
else if (tmpString.ToUpper() == "DUT_P4") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P4;
else if (tmpString.ToUpper() == "DUT_P5") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P5;
else if (tmpString.ToUpper() == "DUT_P6") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P6;
else if (tmpString.ToUpper() == "DUT_P7") caldata.srcSwitchFilterPrimary =
```

```
CalDataRF300.sfPort.DUT_P7;
else if (tmpString.ToUpper() == "DUT_P8") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P8;
else if (tmpString.ToUpper() == "DUT_P1_FWD") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P1_FWD;
else if (tmpString.ToUpper() == "DUT_P2_FWD") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P2_FWD;
else if (tmpString.ToUpper() == "DUT_P3_FWD") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P3_FWD;
else if (tmpString.ToUpper() == "DUT_P4_FWD") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P4_FWD;
else if (tmpString.ToUpper() == "DUT_P5_FWD") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P5_FWD;
else if (tmpString.ToUpper() == "DUT_P6_FWD") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P6_FWD;
else if (tmpString.ToUpper() == "DUT_P7_FWD") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P7_FWD;
else if (tmpString.ToUpper() == "DUT_P8_FWD") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P8_FWD;
else if (tmpString.ToUpper() == "DUT_P1_REV") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P1_REV;
else if (tmpString.ToUpper() == "DUT_P2_REV") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P2_REV;
else if (tmpString.ToUpper() == "DUT_P3_REV") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P3_REV;
else if (tmpString.ToUpper() == "DUT_P4_REV") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P4_REV;
else if (tmpString.ToUpper() == "DUT_P5_REV") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P5_REV;
else if (tmpString.ToUpper() == "DUT_P6_REV") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P6_REV;
else if (tmpString.ToUpper() == "DUT_P7_REV") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P7_REV;
else if (tmpString.ToUpper() == "DUT_P8_REV") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P8_REV;
else if (tmpString.ToUpper() == "DUT_P1_P4_COUPLED") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P1_P4_COUPLED;
else if (tmpString.ToUpper() == "DUT_P5_P8_COUPLED") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P5_P8_COUPLED;
else if (tmpString.ToUpper() == "BB1") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.BB1;
else if (tmpString.ToUpper() == "BB2") caldata.srcSwitchFilterPrimary =
```

```
CalDataRF300.sfPort.BB2;
else if (tmpString.ToUpper() == "BB3") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.BB3;
else if (tmpString.ToUpper() == "BB4") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.BB4;
else if (tmpString.ToUpper() == "HB1_HPF") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.HB1_HPF;
else if (tmpString.ToUpper() == "HB1_LPF") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.HB1_LPF;
else if (tmpString.ToUpper() == "LB1_HPF") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.LB1_HPF;
else if (tmpString.ToUpper() == "LB1_LPF") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.LB1_LPF;
else if (tmpString.ToUpper() == "HB2_HPF") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.HB2_HPF;
else if (tmpString.ToUpper() == "HB2_LPF") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.HB2_LPF;
else if (tmpString.ToUpper() == "LB2_HPF") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.LB2_HPF;
else if (tmpString.ToUpper() == "LB2_LPF") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.LB2_LPF;
else if (tmpString.ToUpper() == "DM1") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DM1;
else if (tmpString.ToUpper() == "DM2") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DM2;
else if (tmpString.ToUpper() == "DM3") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DM3;
else if (tmpString.ToUpper() == "DM4") caldata.srcSwitchFilterPrimary =
CalDataRF300.sfPort.DM4;
else caldata.srcSwitchFilterPrimary = CalDataRF300.sfPort.None;

// Source MT-RF300 Secondary
tmpString = line.Split(',')[5];
if (tmpString.ToUpper() == "DUT_P1") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P1;
else if (tmpString.ToUpper() == "DUT_P2") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P2;
else if (tmpString.ToUpper() == "DUT_P3") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P3;
else if (tmpString.ToUpper() == "DUT_P4") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P4;
else if (tmpString.ToUpper() == "DUT_P5") caldata.srcSwitchFilterSecondary =
```

```
CalDataRF300.sfPort.DUT_P5;
else if (tmpString.ToUpper() == "DUT_P6") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P6;
else if (tmpString.ToUpper() == "DUT_P7") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P7;
else if (tmpString.ToUpper() == "DUT_P8") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P8;
else if (tmpString.ToUpper() == "DUT_P1_FWD") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P1_FWD;
else if (tmpString.ToUpper() == "DUT_P2_FWD") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P2_FWD;
else if (tmpString.ToUpper() == "DUT_P3_FWD") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P3_FWD;
else if (tmpString.ToUpper() == "DUT_P4_FWD") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P4_FWD;
else if (tmpString.ToUpper() == "DUT_P5_FWD") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P5_FWD;
else if (tmpString.ToUpper() == "DUT_P6_FWD") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P6_FWD;
else if (tmpString.ToUpper() == "DUT_P7_FWD") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P7_FWD;
else if (tmpString.ToUpper() == "DUT_P8_FWD") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P8_FWD;
else if (tmpString.ToUpper() == "DUT_P1_REV") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P1_REV;
else if (tmpString.ToUpper() == "DUT_P2_REV") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P2_REV;
else if (tmpString.ToUpper() == "DUT_P3_REV") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P3_REV;
else if (tmpString.ToUpper() == "DUT_P4_REV") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P4_REV;
else if (tmpString.ToUpper() == "DUT_P5_REV") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P5_REV;
else if (tmpString.ToUpper() == "DUT_P6_REV") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P6_REV;
else if (tmpString.ToUpper() == "DUT_P7_REV") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P7_REV;
else if (tmpString.ToUpper() == "DUT_P8_REV") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P8_REV;
else if (tmpString.ToUpper() == "DUT_P1_P4_COUPLED") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P1_P4_COUPLED;
else if (tmpString.ToUpper() == "DUT_P5_P8_COUPLED") caldata.srcSwitchFilterSecondary =
```

```
CalDataRF300.sfPort.DUT_P5_P8_COUPLED;
else if (tmpString.ToUpper() == "BB1") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.BB1;
else if (tmpString.ToUpper() == "BB2") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.BB2;
else if (tmpString.ToUpper() == "BB3") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.BB3;
else if (tmpString.ToUpper() == "BB4") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.BB4;
else if (tmpString.ToUpper() == "HB1_HPF") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.HB1_HPF;
else if (tmpString.ToUpper() == "HB1_LPF") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.HB1_LPF;
else if (tmpString.ToUpper() == "LB1_HPF") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.LB1_HPF;
else if (tmpString.ToUpper() == "LB1_LPF") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.LB1_LPF;
else if (tmpString.ToUpper() == "HB2_HPF") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.HB2_HPF;
else if (tmpString.ToUpper() == "HB2_LPF") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.HB2_LPF;
else if (tmpString.ToUpper() == "LB2_HPF") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.LB2_HPF;
else if (tmpString.ToUpper() == "LB2_LPF") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.LB2_LPF;
else if (tmpString.ToUpper() == "DM1") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DM1;
else if (tmpString.ToUpper() == "DM2") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DM2;
else if (tmpString.ToUpper() == "DM3") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DM3;
else if (tmpString.ToUpper() == "DM4") caldata.srcSwitchFilterSecondary =
CalDataRF300.sfPort.DM4;
else caldata.srcSwitchFilterSecondary = CalDataRF300.sfPort.None;

// Source Coupled Port
tmpString = line.Split(',')[6];
if (tmpString.ToUpper() == "DUT_P1_FWD") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P1_FWD;
else if (tmpString.ToUpper() == "DUT_P2_FWD") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P2_FWD;
else if (tmpString.ToUpper() == "DUT_P3_FWD") caldata.srcCouplerPort =
```

```
CalDataRF300.sfCouplerPort.DUT_P3_FWD;
else if (tmpString.ToUpper() == "DUT_P4_FWD") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P4_FWD;
else if (tmpString.ToUpper() == "DUT_P5_FWD") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P5_FWD;
else if (tmpString.ToUpper() == "DUT_P6_FWD") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P6_FWD;
else if (tmpString.ToUpper() == "DUT_P7_FWD") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P7_FWD;
else if (tmpString.ToUpper() == "DUT_P8_FWD") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P8_FWD;
else if (tmpString.ToUpper() == "DUT_P1_REV") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P1_REV;
else if (tmpString.ToUpper() == "DUT_P2_REV") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P2_REV;
else if (tmpString.ToUpper() == "DUT_P3_REV") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P3_REV;
else if (tmpString.ToUpper() == "DUT_P4_REV") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P4_REV;
else if (tmpString.ToUpper() == "DUT_P5_REV") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P5_REV;
else if (tmpString.ToUpper() == "DUT_P6_REV") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P6_REV;
else if (tmpString.ToUpper() == "DUT_P7_REV") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P7_REV;
else if (tmpString.ToUpper() == "DUT_P8_REV") caldata.srcCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P8_REV;
else caldata.srcCouplerPort = CalDataRF300.sfCouplerPort.None;

// Measure Source Cal Path
tmpString = line.Split(',')[7];
if (tmpString.ToUpper() == "Y" || tmpString.ToUpper() == "X" || tmpString.ToUpper() == "YES")
caldata.measCalPath = true;

// Measure MT-RF300 Primary
tmpString = line.Split(',')[8];
if (tmpString.ToUpper() == "DUT_P1") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P1;
else if (tmpString.ToUpper() == "DUT_P2") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P2;
else if (tmpString.ToUpper() == "DUT_P3") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P3;
```

```
else if (tmpString.ToUpper() == "DUT_P4") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P4;
else if (tmpString.ToUpper() == "DUT_P5") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P5;
else if (tmpString.ToUpper() == "DUT_P6") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P6;
else if (tmpString.ToUpper() == "DUT_P7") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P7;
else if (tmpString.ToUpper() == "DUT_P8") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P8;
else if (tmpString.ToUpper() == "DUT_P1_FWD") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P1_FWD;
else if (tmpString.ToUpper() == "DUT_P2_FWD") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P2_FWD;
else if (tmpString.ToUpper() == "DUT_P3_FWD") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P3_FWD;
else if (tmpString.ToUpper() == "DUT_P4_FWD") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P4_FWD;
else if (tmpString.ToUpper() == "DUT_P5_FWD") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P5_FWD;
else if (tmpString.ToUpper() == "DUT_P6_FWD") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P6_FWD;
else if (tmpString.ToUpper() == "DUT_P7_FWD") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P7_FWD;
else if (tmpString.ToUpper() == "DUT_P8_FWD") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P8_FWD;
else if (tmpString.ToUpper() == "DUT_P1_REV") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P1_REV;
else if (tmpString.ToUpper() == "DUT_P2_REV") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P2_REV;
else if (tmpString.ToUpper() == "DUT_P3_REV") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P3_REV;
else if (tmpString.ToUpper() == "DUT_P4_REV") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P4_REV;
else if (tmpString.ToUpper() == "DUT_P5_REV") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P5_REV;
else if (tmpString.ToUpper() == "DUT_P6_REV") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P6_REV;
else if (tmpString.ToUpper() == "DUT_P7_REV") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P7_REV;
else if (tmpString.ToUpper() == "DUT_P8_REV") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P8_REV;
```

```
else if (tmpString.ToUpper() == "DUT_P1_P4_COUPLED") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P1_P4_COUPLED;
else if (tmpString.ToUpper() == "DUT_P5_P8_COUPLED") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DUT_P5_P8_COUPLED;
else if (tmpString.ToUpper() == "BB1") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.BB1;
else if (tmpString.ToUpper() == "BB2") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.BB2;
else if (tmpString.ToUpper() == "BB3") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.BB3;
else if (tmpString.ToUpper() == "BB4") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.BB4;
else if (tmpString.ToUpper() == "HB1_HPF") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.HB1_HPF;
else if (tmpString.ToUpper() == "HB1_LPF") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.HB1_LPF;
else if (tmpString.ToUpper() == "LB1_HPF") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.LB1_HPF;
else if (tmpString.ToUpper() == "LB1_LPF") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.LB1_LPF;
else if (tmpString.ToUpper() == "HB2_HPF") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.HB2_HPF;
else if (tmpString.ToUpper() == "HB2_LPF") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.HB2_LPF;
else if (tmpString.ToUpper() == "LB2_HPF") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.LB2_HPF;
else if (tmpString.ToUpper() == "LB2_LPF") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.LB2_LPF;
else if (tmpString.ToUpper() == "DM1") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DM1;
else if (tmpString.ToUpper() == "DM2") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DM2;
else if (tmpString.ToUpper() == "DM3") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DM3;
else if (tmpString.ToUpper() == "DM4") caldata.measSwitchFilterPrimary =
CalDataRF300.sfPort.DM4;
else caldata.measSwitchFilterPrimary = CalDataRF300.sfPort.None;

// Measure MT-RF300 Secondary
tmpString = line.Split(',')[9];
if (tmpString.ToUpper() == "DUT_P1") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P1;
```

```
else if (tmpString.ToUpper() == "DUT_P2") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P2;
else if (tmpString.ToUpper() == "DUT_P3") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P3;
else if (tmpString.ToUpper() == "DUT_P4") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P4;
else if (tmpString.ToUpper() == "DUT_P5") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P5;
else if (tmpString.ToUpper() == "DUT_P6") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P6;
else if (tmpString.ToUpper() == "DUT_P7") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P7;
else if (tmpString.ToUpper() == "DUT_P8") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P8;
else if (tmpString.ToUpper() == "DUT_P1_FWD") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P1_FWD;
else if (tmpString.ToUpper() == "DUT_P2_FWD") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P2_FWD;
else if (tmpString.ToUpper() == "DUT_P3_FWD") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P3_FWD;
else if (tmpString.ToUpper() == "DUT_P4_FWD") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P4_FWD;
else if (tmpString.ToUpper() == "DUT_P5_FWD") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P5_FWD;
else if (tmpString.ToUpper() == "DUT_P6_FWD") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P6_FWD;
else if (tmpString.ToUpper() == "DUT_P7_FWD") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P7_FWD;
else if (tmpString.ToUpper() == "DUT_P8_FWD") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P8_FWD;
else if (tmpString.ToUpper() == "DUT_P1_REV") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P1_REV;
else if (tmpString.ToUpper() == "DUT_P2_REV") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P2_REV;
else if (tmpString.ToUpper() == "DUT_P3_REV") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P3_REV;
else if (tmpString.ToUpper() == "DUT_P4_REV") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P4_REV;
else if (tmpString.ToUpper() == "DUT_P5_REV") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P5_REV;
else if (tmpString.ToUpper() == "DUT_P6_REV") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P6_REV;
```

```
else if (tmpString.ToUpper() == "DUT_P7_REV") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P7_REV;
else if (tmpString.ToUpper() == "DUT_P8_REV") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P8_REV;
else if (tmpString.ToUpper() == "DUT_P1_P4_COUPLED") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P1_P4_COUPLED;
else if (tmpString.ToUpper() == "DUT_P5_P8_COUPLED") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DUT_P5_P8_COUPLED;
else if (tmpString.ToUpper() == "BB1") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.BB1;
else if (tmpString.ToUpper() == "BB2") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.BB2;
else if (tmpString.ToUpper() == "BB3") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.BB3;
else if (tmpString.ToUpper() == "BB4") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.BB4;
else if (tmpString.ToUpper() == "HB1_HPF") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.HB1_HPF;
else if (tmpString.ToUpper() == "HB1_LPF") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.HB1_LPF;
else if (tmpString.ToUpper() == "LB1_HPF") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.LB1_HPF;
else if (tmpString.ToUpper() == "LB1_LPF") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.LB1_LPF;
else if (tmpString.ToUpper() == "HB2_HPF") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.HB2_HPF;
else if (tmpString.ToUpper() == "HB2_LPF") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.HB2_LPF;
else if (tmpString.ToUpper() == "LB2_HPF") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.LB2_HPF;
else if (tmpString.ToUpper() == "LB2_LPF") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.LB2_LPF;
else if (tmpString.ToUpper() == "DM1") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DM1;
else if (tmpString.ToUpper() == "DM2") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DM2;
else if (tmpString.ToUpper() == "DM3") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DM3;
else if (tmpString.ToUpper() == "DM4") caldata.measSwitchFilterSecondary =
CalDataRF300.sfPort.DM4;
else caldata.measSwitchFilterSecondary = CalDataRF300.sfPort.None;
```

```
// Measure Port
tmpString = line.Split(',')[10];
if (tmpString.ToUpper() == "P1") caldata.measPort = MeasPort.P1;
else if (tmpString.ToUpper() == "P2") caldata.measPort = MeasPort.P2;
else if (tmpString.ToUpper() == "P3") caldata.measPort = MeasPort.P3;
else if (tmpString.ToUpper() == "P4") caldata.measPort = MeasPort.P4;
else if (tmpString.ToUpper() == "P5") caldata.measPort = MeasPort.P5;
else if (tmpString.ToUpper() == "P6") caldata.measPort = MeasPort.P6;
else if (tmpString.ToUpper() == "P7") caldata.measPort = MeasPort.P7;
else if (tmpString.ToUpper() == "P8") caldata.measPort = MeasPort.P8;
else if (tmpString.ToUpper() == "P9") caldata.measPort = MeasPort.P9;
else if (tmpString.ToUpper() == "P10") caldata.measPort = MeasPort.P10;
else if (tmpString.ToUpper() == "P11") caldata.measPort = MeasPort.P11;
else if (tmpString.ToUpper() == "P12") caldata.measPort = MeasPort.P12;
else if (tmpString.ToUpper() == "P13") caldata.measPort = MeasPort.P13;
else if (tmpString.ToUpper() == "P14") caldata.measPort = MeasPort.P14;
else if (tmpString.ToUpper() == "P15") caldata.measPort = MeasPort.P15;
else if (tmpString.ToUpper() == "P16") caldata.measPort = MeasPort.P16;
else if (tmpString.ToUpper() == "P1_FWD") caldata.measPort = MeasPort.P1_Fwd;
else if (tmpString.ToUpper() == "P2_FWD") caldata.measPort = MeasPort.P2_Fwd;
else if (tmpString.ToUpper() == "P3_FWD") caldata.measPort = MeasPort.P3_Fwd;
else if (tmpString.ToUpper() == "P4_FWD") caldata.measPort = MeasPort.P4_Fwd;
else if (tmpString.ToUpper() == "P5_FWD") caldata.measPort = MeasPort.P5_Fwd;
else if (tmpString.ToUpper() == "P6_FWD") caldata.measPort = MeasPort.P6_Fwd;
else if (tmpString.ToUpper() == "P7_FWD") caldata.measPort = MeasPort.P7_Fwd;
else if (tmpString.ToUpper() == "P8_FWD") caldata.measPort = MeasPort.P8_Fwd;
else if (tmpString.ToUpper() == "P1_REV") caldata.measPort = MeasPort.P1_Rev;
else if (tmpString.ToUpper() == "P2_REV") caldata.measPort = MeasPort.P2_Rev;
else if (tmpString.ToUpper() == "P3_REV") caldata.measPort = MeasPort.P3_Rev;
else if (tmpString.ToUpper() == "P4_REV") caldata.measPort = MeasPort.P4_Rev;
else if (tmpString.ToUpper() == "P5_REV") caldata.measPort = MeasPort.P5_Rev;
else if (tmpString.ToUpper() == "P6_REV") caldata.measPort = MeasPort.P6_Rev;
else if (tmpString.ToUpper() == "P7_REV") caldata.measPort = MeasPort.P7_Rev;
else if (tmpString.ToUpper() == "P8_REV") caldata.measPort = MeasPort.P8_Rev;
else if (tmpString.ToUpper() == "CAL") caldata.measPort = MeasPort.CAL;
else if (tmpString.ToUpper() == "M1") caldata.measPort = MeasPort.M1;
else if (tmpString.ToUpper() == "M2") caldata.measPort = MeasPort.M2;
else if (tmpString.ToUpper() == "M3") caldata.measPort = MeasPort.M3;
else if (tmpString.ToUpper() == "M4") caldata.measPort = MeasPort.M4;
else if (tmpString.ToUpper() == "M5") caldata.measPort = MeasPort.M5;
else if (tmpString.ToUpper() == "M6") caldata.measPort = MeasPort.M6;
else if (tmpString.ToUpper() == "M7") caldata.measPort = MeasPort.M7;
```

```
else if (tmpString.ToUpper() == "M8") caldata.measPort = MeasPort.M8;
else if (tmpString.ToUpper() == "M9") caldata.measPort = MeasPort.M9;
else if (tmpString.ToUpper() == "M10") caldata.measPort = MeasPort.M10;
else if (tmpString.ToUpper() == "M11") caldata.measPort = MeasPort.M11;
else if (tmpString.ToUpper() == "M12") caldata.measPort = MeasPort.M12;
else if (tmpString.ToUpper() == "M13") caldata.measPort = MeasPort.M13;
else if (tmpString.ToUpper() == "M14") caldata.measPort = MeasPort.M14;
else if (tmpString.ToUpper() == "M15") caldata.measPort = MeasPort.M15;
else if (tmpString.ToUpper() == "M16") caldata.measPort = MeasPort.M16;
else if (tmpString.ToUpper() == "M17") caldata.measPort = MeasPort.M17;
else if (tmpString.ToUpper() == "M18") caldata.measPort = MeasPort.M18;
else if (tmpString.ToUpper() == "M19") caldata.measPort = MeasPort.M19;
else if (tmpString.ToUpper() == "M20") caldata.measPort = MeasPort.M20;
else if (tmpString.ToUpper() == "M21") caldata.measPort = MeasPort.M21;
else if (tmpString.ToUpper() == "M22") caldata.measPort = MeasPort.M22;
else if (tmpString.ToUpper() == "M23") caldata.measPort = MeasPort.M23;
else if (tmpString.ToUpper() == "M24") caldata.measPort = MeasPort.M24;
else if (tmpString.ToUpper() == "M25") caldata.measPort = MeasPort.M25;
else if (tmpString.ToUpper() == "M26") caldata.measPort = MeasPort.M26;
else caldata.measPort = MeasPort.NONE;

// Measure Port Alias
caldata.measPortAlias = line.Split(',')[11];

// Measure Coupled Port
tmpString = line.Split(',')[12];
if (tmpString.ToUpper() == "DUT_P1_FWD") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P1_FWD;
else if (tmpString.ToUpper() == "DUT_P2_FWD") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P2_FWD;
else if (tmpString.ToUpper() == "DUT_P3_FWD") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P3_FWD;
else if (tmpString.ToUpper() == "DUT_P4_FWD") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P4_FWD;
else if (tmpString.ToUpper() == "DUT_P5_FWD") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P5_FWD;
else if (tmpString.ToUpper() == "DUT_P6_FWD") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P6_FWD;
else if (tmpString.ToUpper() == "DUT_P7_FWD") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P7_FWD;
else if (tmpString.ToUpper() == "DUT_P8_FWD") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P8_FWD;
```

```
else if (tmpString.ToUpper() == "DUT_P1_REV") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P1_REV;
else if (tmpString.ToUpper() == "DUT_P2_REV") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P2_REV;
else if (tmpString.ToUpper() == "DUT_P3_REV") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P3_REV;
else if (tmpString.ToUpper() == "DUT_P4_REV") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P4_REV;
else if (tmpString.ToUpper() == "DUT_P5_REV") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P5_REV;
else if (tmpString.ToUpper() == "DUT_P6_REV") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P6_REV;
else if (tmpString.ToUpper() == "DUT_P7_REV") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P7_REV;
else if (tmpString.ToUpper() == "DUT_P8_REV") caldata.measCouplerPort =
CalDataRF300.sfCouplerPort.DUT_P8_REV;
else caldata.measCouplerPort = CalDataRF300.sfCouplerPort.None;

// Frequency / Level
caldata.srcFreq = double.Parse(line.Split(',')[13]);
caldata.srcLevel = double.Parse(line.Split(',')[14]);

// Filter
tmpString = line.Split(',')[15];
if (tmpString.ToUpper() == "BYPASS") caldata.measFilter = MeasFilt.Bypass;
else if (tmpString.ToUpper() == "ATTEN") caldata.measFilter = MeasFilt.Atten;
else if (tmpString.ToUpper() == "F1" || tmpString.ToUpper() == "FILT1") caldata.measFilter =
MeasFilt.FILT1;
else if (tmpString.ToUpper() == "F2" || tmpString.ToUpper() == "FILT2") caldata.measFilter =
MeasFilt.FILT2;
else if (tmpString.ToUpper() == "F3" || tmpString.ToUpper() == "FILT3") caldata.measFilter =
MeasFilt.FILT3;
else caldata.measFilter = MeasFilt.Bypass; // Default

caldata.measAtten = double.Parse(line.Split(',')[16]);
caldata.modulationType = line.Split(',')[17];
caldata.modulationFile = line.Split(',')[18];
caldata.dutyCycle = double.Parse(line.Split(',')[19]);
caldata.comment = line.Split(',')[20];

caldatum.Add(caldata);
}
```

```
}
}
}
catch (FileNotFoundException error)
{
Console.WriteLine("\n{0}", error.Message);
}

return Tuple.Create(caldatum, info, CalSelect, coupledPort, switchSelect, attenExternal);
}
}
}
```

Enums.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MT.TestStudio.Enums
{
public enum Direction { FWD, REV };
public enum SigGenV3 { SG1, SG2, SG1_SG2, NOISE, ISOLATE };
public enum SigGen { SG1, SG2, SG1_SG2, NOISE, ISOLATE, VNA_1P, VNA_2P };
public enum SrcPort { P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16,
SG1_OUT, CAL, NONE };
public enum MeasPort { P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16,
P1_Fwd, P1_Rev, P2_Fwd, P2_Rev, P3_Fwd, P3_Rev, P4_Fwd, P4_Rev, P5_Fwd, P5_Rev,
P6_Fwd, P6_Rev, P7_Fwd, P7_Rev, P8_Fwd, P8_Rev, M1, M2, M3, M4, M5, M6, M7, M8, M9,
M10, M11, M12, M13, M14, M15, M16, M17, M18, M19, M20, M21, M22, M23, M24, M25, M26,
MA, MB, MC, MD, SRC, CAL, NONE };
public enum SrcAmp { A1_SRC1, A1_SRC2, A2, A3, A4_MEAS, All };
public enum MeasAmp { A1, A2, All };
public enum SrcAmpEnable { All, None, Hold, A1, A2, A3, A1_A2, A1_A3, A2_A3 };
public enum MeasAmpEnable { All, None, Hold, A1, A2 };
public enum AmpState { Enable, Bypass, Hold };
public enum LNA { SRC1LNA1, SRC2LNA1, SRCLNA2, SRCLNA3, SRCMEAS, MEASLNA1,
MEASLNA2 };
public enum MeasAtten { Ref, A_p25dB, A_p5dB, A_1dB, A_2dB, A_4dB, A_8dB, A_16dB,
A_32dB }
public enum MeasFilt { Atten, Bypass, FILT1, FILT2, FILT3, NONE };
```

```
public enum InputRegisterCommand { NoOperation, WriteToInputRegisterNoUpdate,
UpdateDacRegisterFromInputRegister, WriteAndUpdateDacRegister, WriteToControlRegister,
Reset, DisableDaisyChain, ReadbackInputRegister, ReadbackDacRegister,
ReadbackControlRegister, FullReset };

// Needed for port module calibraion and associated Cal_Import methods
public enum SrcAmpConfig { Bypass, A1, A2, A3, A1_A2, A1_A3, A2_A3, A1_A2_A3, A4_MEAS,
A1_SG2, None };
public enum MeasAmpConfig { Bypass, A1, A2, A1_A2 };
public enum SrcMeasAmpConfig { Bypass, A4_MEAS, A1, A2, A4_MEAS_A1, A4_MEAS_A2,
A1_A2, A4_MEAS_A1_A2 };
public enum CalAmpConfig { Bypass, SA1, SA2, SA3, MA1, MA2, SA1_SA2, SA1_SA3,
SA2_SA3, SA1_SA2_SA3, MA1_MA2, SA1_MA1, SA1_MA2, SA1_MA1_MA2, SA2_MA1,
SA2_MA2, SA2_MA1_MA2, SA3_MA1, SA3_MA2, SA3_MA1_MA2, SA1_SA2_MA1,
SA1_SA2_MA2, SA1_SA2_MA1_MA2, SA1_SA3_MA1, SA1_SA3_MA2, SA1_SA3_MA1_MA2,
SA2_SA3_MA1, SA2_SA3_MA2, SA2_SA3_MA1_MA2, SA1_SA2_SA3_MA1,
SA1_SA2_SA3_MA2, SA1_SA2_SA3_MA1_MA2 };

public enum Vsupply { V_m2p5, V_m3p3, V_3p3, V_5A, V_5B };

//enums
public enum MeasChanMode { V, I }
public enum Gain { G1, G10, G100, G1000 };
public enum HMeasMode { LowC, HighC };
public enum LMeasMode { LowC, HighC };
public enum LMode { Vsource, Isource };
public enum HRange { R_3_5A, R_2A, R_1A, R_500mA, R_200mA, R_100mA, R_50mA,
R_20mA, R_10mA, R_5mA, R_1mA, R_100uA, R_10uA, R_10mA_1K, R_5mA_1K };
public enum LRange { R_200mA, R_100mA, R_35mA, R_10mA, R_1mA, R_100uA, R_10uA,
R_1uA, R_100nA, R_10nA };
public enum Atten { A_0dB, A_2dB, A_4dB, A_6dB, A_8dB, A_10dB, A_12dB, A_14dB, A_16dB,
A_18dB, A_20dB, A_22dB, A_24dB, A_26dB, A_28dB, A_30dB };
public enum chanType { lowPI, lowPV, highP, front, back, temp }


public enum ModulationType { None, CW, GSM, EDGE, LTE_FDD, LTE_TDD, TDSCDMA,
WCDMA, WLAN };
public enum ChannelBandwidth { OnePointFourMHz, ThreeMHz, FiveMHz, TenMHz, FifteenMHz,
TwentyMHz, NA };

public enum YesNo
{
```

```
Yes,
No
}
}
```

Exceptions.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MT.TestStudio.Exceptions
{
/// <summary>
/// Initializes a new instance of the ProjectFileIOException class with default
/// properties.
/// </summary>
public class ProjectFileIOException : Exception
{
/// <summary>
/// Initializes a new instance of the ProjectFileIOException class with default
/// properties.
/// </summary>
public ProjectFileIOException()
{
}

/// <summary>
/// Initializes a new instance of the ProjectFileIOException class with a
/// specified error message.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
public ProjectFileIOException(string message)
: base(message)
{
}

/// <summary>
/// Initializes a new instance of the ProjectFileIOException class with a
/// specified error message and a reference to the inner exception that is the cause
```

```
//  of this exception.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
/// <param name="inner">The exception that is the cause of the current exception. If
/// the inner parameter is not null, the current exception is raised in a catch block
/// that handles the inner exception.</param>
public ProjectFileIOException(string message, Exception inner)
: base(message, inner)
{
}
}

/// <summary>
/// Initializes a new instance of the ProductConfigurationException class with default
/// properties.
/// </summary>
public class ProductConfigurationException : Exception
{
/// <summary>
/// Initializes a new instance of the ProductConfigurationException class with default
/// properties.
/// </summary>
public ProductConfigurationException()
{
}

/// <summary>
/// Initializes a new instance of the ProductConfigurationException class with a
/// specified error message.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
public ProductConfigurationException(string message)
: base(message)
{
}

/// <summary>
/// Initializes a new instance of the ProductConfigurationException class with a
/// specified error message and a reference to the inner exception that is the cause
//  of this exception.
```

```
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
/// <param name="inner">The exception that is the cause of the current exception. If
/// the inner parameter is not null, the current exception is raised in a catch block
/// that handles the inner exception.</param>
public ProductConfigurationException(string message, Exception inner)
: base(message, inner)
{
}
}

/// <summary>
/// Initializes a new instance of the ProjectConfigurationException class with default
/// properties.
/// </summary>
public class ProjectConfigurationException : Exception
{
/// <summary>
/// Initializes a new instance of the ProjectConfigurationException class with default
/// properties.
/// </summary>
public ProjectConfigurationException()
{
}

/// <summary>
/// Initializes a new instance of the ProjectConfigurationException class with a
/// specified error message.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
public ProjectConfigurationException(string message)
: base(message)
{
}

/// <summary>
/// Initializes a new instance of the ProjectConfigurationException class with a
/// specified error message and a reference to the inner exception that is the cause
//  of this exception.
/// </summary>
```

```
/// <param name="message">The error message that explains the reason for the
exception.</param>
/// <param name="inner">The exception that is the cause of the current exception. If
/// the inner parameter is not null, the current exception is raised in a catch block
/// that handles the inner exception.</param>
public ProjectConfigurationException(string message, Exception inner)
: base(message, inner)
{
}
}

/// <summary>
/// Initializes a new instance of the TestConditionException class with default
/// properties.
/// </summary>
public class TestConditionException : Exception
{
/// <summary>
/// Initializes a new instance of the TestConditionException class with default
/// properties.
/// </summary>
public TestConditionException()
{
}

/// <summary>
/// Initializes a new instance of the TestConditionException class with a
/// specified error message.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
public TestConditionException(string message)
: base(message)
{
}

/// <summary>
/// Initializes a new instance of the TestConditionException class with a
/// specified error message and a reference to the inner exception that is the cause
//  of this exception.
/// </summary>
/// <param name="message">The error message that explains the reason for the
```

```
exception.</param>
/// <param name="inner">The exception that is the cause of the current exception. If
/// the inner parameter is not null, the current exception is raised in a catch block
/// that handles the inner exception.</param>
public TestConditionException(string message, Exception inner)
: base(message, inner)
{
}
}

/// <summary>
/// Initializes a new instance of the CalibrationResultsException class with default
/// properties.
/// </summary>
public class CalibrationResultsException : Exception
{
/// <summary>
/// Initializes a new instance of the CalibrationResultsException class with default
/// properties.
/// </summary>
public CalibrationResultsException()
{
}

/// <summary>
/// Initializes a new instance of the CalibrationResultsException class with a
/// specified error message.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
public CalibrationResultsException(string message)
: base(message)
{
}

/// <summary>
/// Initializes a new instance of the CalibrationResultsException class with a
/// specified error message and a reference to the inner exception that is the cause
//  of this exception.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
```

```
/// <param name="inner">The exception that is the cause of the current exception. If
/// the inner parameter is not null, the current exception is raised in a catch block
/// that handles the inner exception.</param>
public CalibrationResultsException(string message, Exception inner)
: base(message, inner)
{
}
}

/// <summary>
/// Initializes a new instance of the TestLimitException class with default
/// properties.
/// </summary>
public class TestLimitException : Exception
{
/// <summary>
/// Initializes a new instance of the TestLimitException class with default
/// properties.
/// </summary>
public TestLimitException()
{
}

/// <summary>
/// Initializes a new instance of the TestLimitException class with a
/// specified error message.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
public TestLimitException(string message)
: base(message)
{
}

/// <summary>
/// Initializes a new instance of the TestLimitException class with a
/// specified error message and a reference to the inner exception that is the cause
//  of this exception.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
/// <param name="inner">The exception that is the cause of the current exception. If
```

```
/// the inner parameter is not null, the current exception is raised in a catch block
/// that handles the inner exception.</param>
public TestLimitException(string message, Exception inner)
: base(message, inner)
{
}
}

/// <summary>
/// Initializes a new instance of the DigitalException class with default
/// properties.
/// </summary>
public class DigitalException : Exception
{
/// <summary>
/// Initializes a new instance of the DigitalException class with default
/// properties.
/// </summary>
public DigitalException()
{
}

/// <summary>
/// Initializes a new instance of the DigitalException class with a
/// specified error message.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
public DigitalException(string message)
: base(message)
{
}

/// <summary>
/// Initializes a new instance of the DigitalException class with a
/// specified error message and a reference to the inner exception that is the cause
//  of this exception.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
/// <param name="inner">The exception that is the cause of the current exception. If
/// the inner parameter is not null, the current exception is raised in a catch block
```

```
/// that handles the inner exception.</param>
public DigitalException(string message, Exception inner)
: base(message, inner)
{
}
}

/// <summary>
/// Initializes a new instance of the CalibrationDefinitionException class with default
/// properties.
/// </summary>
public class CalibrationDefinitionException : Exception
{
/// <summary>
/// Initializes a new instance of the CalibrationDefinitionException class with default
/// properties.
/// </summary>
public CalibrationDefinitionException()
{
}

/// <summary>
/// Initializes a new instance of the CalibrationDefinitionException class with a
/// specified error message.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
public CalibrationDefinitionException(string message)
: base(message)
{
}

/// <summary>
/// Initializes a new instance of the CalibrationDefinitionException class with a
/// specified error message and a reference to the inner exception that is the cause
//  of this exception.
/// </summary>
/// <param name="message">The error message that explains the reason for the
exception.</param>
/// <param name="inner">The exception that is the cause of the current exception. If
/// the inner parameter is not null, the current exception is raised in a catch block
/// that handles the inner exception.</param>
```

```
public CalibrationDefinitionException(string message, Exception inner)
: base(message, inner)
{
}
}

}
```

ProductConfigModel.cs

```
//using MT.Core.Common;
using MT.TestStudio.Exceptions;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using System.Xml.Linq;
using System.Xml.Schema;

namespace MerlinTest.Tools.TestStudio.Model
{
//    /// <summary>
//    /// Product Config Model class.
//    /// </summary>
//    public class ProductConfigModel
//    {
//        #region Private Member Variables



//        #endregion Private Member Variables

//        #region Constructor



//        #endregion Constructor

//        #region Public Properties
```

```
//        public bool ContinueOnFail { get; set; }
//        public bool StopOnFail { get; set; }
//        public bool ContinueOnAllFail { get; set; }
//        public bool StopOnalarm { get; set; }
//        public bool ContinueOnAlarm { get; set; }
//        public bool GoldUnitEnable { get; set; }
//        public bool UserCalibrationEnable { get; set; }
//        public int CalibrationExpiration { get; set; }
//        public bool OfflineQaAvailable { get; set; }
//        public bool EncryptRequired { get; set; }
//        public int InlineNthDevice { get; set; }
//        public int LogNthDevice { get; set; }
//        public int NumberOfsites { get; set; }

//        #endregion Public Properties

//        #region Public Events / Delegates

//        /// <summary>
//        /// Event of type PropertyChangedEventHandler which is used to implement
INotifyPropertyChanged
//        /// and provide notification of when a property has changed.
//        /// </summary>
//        public event PropertyChangedEventHandler PropertyChanged;

//        #endregion Public Events / Delegates

//        #region Public Methods

//        /// <summary>
//        /// Converts a string into a boolean.
//        /// </summary>
//        /// <param name="Value">Value to convert</param>
//        /// <returns>The converted values.</returns>
//        private static bool ConvertStringToBool(string Value, string ErrorText, int LineNumber)
//        {

//            Value = Value.Trim().ToLower();

//            switch (Value)
//            {
//                case "yes":
```

```
//                case "y":
//                case "true":
//                case "pass":
//                case "p":
//                case "1":
//                    return true;
//                case "false":
//                case "fail":
//                case "0":
//                case "f":
//                case "n":
//                case "no":
//                    return false;
//                default:
//                    throw new ProductConfigurationException(string.Format("{0} - Line {1} Value of {2}
is invalid. (Use Y or N) ", ErrorText, LineNumber, Value));
//            }
//        }

//        /// <summary>
//        /// Converts a string into a boolean.
//        /// </summary>
//        /// <param name="Value">Value to convert</param>
//        /// <param name="ErrorTextr">Error text to return.</param>
//        /// <returns>The converted values.</returns>
//        private static bool ConvertStringToBool(string Value, string ErrorText)
//        {

//            Value = Value.Trim().ToLower();

//            switch (Value)
//            {
//                case "yes":
//                case "y":
//                case "true":
//                case "pass":
//                case "p":
//                case "1":
//                    return true;
//                case "false":
//                case "fail":
//                case "0":
```

```
//                case "f":
//                case "n":
//                case "no":
//                    return false;
//                default:
//                    throw new ProductConfigurationException(string.Format("{0} - Value of {1} is invalid.
(Use True/Yes/Y or False/No/N) ", ErrorText, Value));
//            }
//        }

//        /// <summary>
//        /// Helper Method to convert a string to an int.
//        /// </summary>
//        /// <param name="Value">Value to convert.</param>
//        /// <param name="ErrorText">Error text to display in exception.</param>
//        /// <returns>The converted value.</returns>
//        private static int ConvertStringToInt(string Value, string ErrorText, int LineNumber)
//        {
//            if (int.TryParse(Value, out int result) == true)
//            {
//                return result;
//            }
//            else
//            {
//                throw new ProductConfigurationException(string.Format("{0} - Line {1} Value of {2} is
invalid. ", ErrorText, LineNumber, Value));
//            }
//        }

//        /// <summary>
//        /// Helper Method to convert a string to an int.
//        /// </summary>
//        /// <param name="Value">Value to convert.</param>
//        /// <param name="ErrorText">Error text to display in exception.</param>
//        /// <returns>The converted value.</returns>
//        private static int ConvertStringToInt(string Value, string ErrorText)
//        {
//            if (int.TryParse(Value, out int result) == true)
//            {
//                return result;
//            }
//            else
```

```
//          {
//              throw new ProductConfigurationException(string.Format("{0} - Value of {1} is invalid. ",
ErrorText, Value));
//          }
//      }

//      /// <summary>
//      /// Helper Method to convert a string to a double.
//      /// </summary>
//      /// <param name="Value">Value to convert.</param>
//      /// <param name="ErrorText">Error text to display in exception.</param>
//      /// <returns>The converted value.</returns>
//      private static double ConvertStringToDouble(string Value, string ErrorText, int LineNumber)
//      {
//          if (double.TryParse(Value, out double result) == true)
//          {
//              return result;
//          }
//          else
//          {
//              throw new ProductConfigurationException(string.Format("{0} - Line {1} Value of {2} is
invalid. ", ErrorText, LineNumber, Value));
//          }
//      }

//      /// <summary>
//      /// Method that causes a file to be read in.
//      /// </summary>
//      /// <param name="FileName">The file to be read in.</param>
//      public void LoadDataFile(string FileName)
//      {
//          try
//          {
//              // This method is called by the framework to get the schema for this type.
//              // We return an existing schema from disk.
//              Assembly asmb = Assembly.GetExecutingAssembly();
//              string[] names = asmb.GetManifestResourceNames();
//              Stream stream =
asmb.GetManifestResourceStream("MT.TestStudio.Model.Schemas.ProductConfig.xsd");

//              XmlSchema vsvSchema = XmlSchema.Read(stream, null);
//              XmlSchemaSet schemas = new XmlSchemaSet();
```

```
//              schemas.Add(vsvSchema);
//              XDocument document = XDocument.Load(FileName);
//              string msg = "";
//              document.Validate(schemas, (o, e) =>
//              {
//                  msg += e.Message + "(Line " + e.Exception.LineNumber + ") " +
Environment.NewLine;
//              });

//              if (msg == string.Empty)
//              {
//                  int fileVersion =
HelperMethods.ConvertStringToIntSafely(document.Root.Attribute("Version").Value);

//                  if (fileVersion == 1)
//                  {

//                      // Work out what the namespace is for the XML file.
//                      XNamespace productConfigNS = document.Root.Name.Namespace;

//                      var result = document.Descendants(productConfigNS + "Settings").Select(cd =>
new
//                      {
//                          ContinueOnFail = cd.Element(productConfigNS + "ContinueOnFail").Value,
//                          StopOnFail = cd.Element(productConfigNS + "StopOnFail").Value,
//                          ContinueOnAllFail = cd.Element(productConfigNS +
"ContinueOnAllFail").Value,
//                          StopOnAlarm = cd.Element(productConfigNS + "StopOnAlarm").Value,
//                          GoldUnitEnable = cd.Element(productConfigNS + "GoldUnitEnable").Value,
//                          UserCalibration = cd.Element(productConfigNS + "UserCalibration").Value,
//                          CalibrationExpiration = cd.Element(productConfigNS +
"CalibrationExpiration").Value,
//                          OfflineQAEnable = cd.Element(productConfigNS + "OfflineQAEnable").Value,
//                          EncryptRequired = cd.Element(productConfigNS + "EncryptRequired").Value,
//                          InlineEnabledNthDevice = cd.Element(productConfigNS +
"InlineEnabledNthDevice").Value,
//                          LogNthDevice = cd.Element(productConfigNS + "LogNthDevice").Value,
//                          NumberOfSites = cd.Element(productConfigNS + "NumberOfSites").Value,
//                      });

//                      foreach (var cd in result)
```

```
//                  {
//                      this.ContinueOnFail = bool.Parse(cd.ContinueOnFail);
//                      this.StopOnFail = bool.Parse(cd.StopOnFail);
//                      this.ContinueOnAllFail = bool.Parse(cd.ContinueOnAllFail);
//                      this.StopOnalarm = bool.Parse(cd.StopOnAlarm);
//                      this.GoldUnitEnable = bool.Parse(cd.GoldUnitEnable);
//                      this.UserCalibrationEnable = bool.Parse(cd.UserCalibration);
//                      this.CalibrationExpiration = int.Parse(cd.CalibrationExpiration);
//                      this.OfflineQaAvailable = bool.Parse(cd.OfflineQAEnable);
//                      this.EncryptRequired = bool.Parse(cd.EncryptRequired);
//                      this.InlineNthDevice = int.Parse(cd.InlineEnabledNthDevice);
//                      this.LogNthDevice = int.Parse(cd.LogNthDevice);
//                      this.NumberOfsites = int.Parse(cd.NumberOfSites);
//                  }
//              }
//              else
//              {

//              }
//          }
//          else
//          {
//              throw new ProductConfigurationException("Document invalid: " + msg);
//          }

//          // Notify the View-Model that the product config data has been read in.
//          OnPropertyChanged("CsvDataLoaded");
//      }
//      catch (Exception ex)
//      {
//          string message = string.Format("An error occured while loading a product configuration
file {0}. {1}", FileName, ex.Message);
//          TestStudioModel.trace.TraceError(message);
//          throw new ProductConfigurationException(message);
//      }
//  }

//  /// <summary>
//  /// Method that causes a file to be read in.
//  /// </summary>
//  /// <param name="FileName">The file to be read in.</param>
//  public void LoadCsvDataFile(string FileName)
```

```
//       {
//           try
//           {
//               // Read header of file and split by ','
//               var headerInfo = (from lines in File.ReadLines(FileName)
//                                 let columns = lines.Split(',')
//                                 select columns).ToList();

//               // Meta Data Stored in Header
//               for (int index = 0; index < headerInfo.Count; index++)
//               {
//                   if (headerInfo[index].Length > 1)
//                   {
//                       if (headerInfo[index][0].ToUpper() == "CONTINUE ON FAIL")
//                       {
//                           this.ContinueOnFail = ConvertStringToBool(headerInfo[index][1],
headerInfo[index][0], index);
//                       }

//                       if (headerInfo[index][0].ToUpper() == "STOP ON FAIL")
//                       {
//                           this.StopOnFail = ConvertStringToBool(headerInfo[index][1],
headerInfo[index][0], index);
//                       }

//                       if (headerInfo[index][0].ToUpper() == "CONTINUE ON ALL FAIL")
//                       {
//                           this.ContinueOnAllFail = ConvertStringToBool(headerInfo[index][1],
headerInfo[index][0], index);
//                       }

//                       if (headerInfo[index][0].ToUpper() == "STOP ON ALARM")
//                       {
//                           this.StopOnalarm = ConvertStringToBool(headerInfo[index][1],
headerInfo[index][0], index);
//                       }

//                       if (headerInfo[index][0].ToUpper() == "CONTINUE ON ALARM")
//                       {
//                           this.ContinueOnAlarm = ConvertStringToBool(headerInfo[index][1],
headerInfo[index][0], index);
//                       }
```

```
//                  if (headerInfo[index][0].ToUpper() == "GOLD UNIT ENABLE/DISABLE")
//                  {
//                      this.GoldUnitEnable = ConvertStringToBool(headerInfo[index][1],
headerInfo[index][0], index);
//                  }

//                  if (headerInfo[index][0].ToUpper() == "USER CALIBRATION")
//                  {
//                      this.UserCalibrationEnable = ConvertStringToBool(headerInfo[index][1],
headerInfo[index][0], index);
//                  }

//                  if (headerInfo[index][0].ToUpper() == "CALIBRATION EXPIRATION")
//                  {
//                      this.CalibrationExpiration = ConvertStringToInt(headerInfo[index][1],
headerInfo[index][0], index);
//                  }

//                  if (headerInfo[index][0].ToUpper() == "OFFLINE QA ENABLE")
//                  {
//                      this.OfflineQaAvailable = ConvertStringToBool(headerInfo[index][1],
headerInfo[index][0], index);
//                  }

//                  if (headerInfo[index][0].ToUpper() == "ENCRYPT REQUIRED")
//                  {
//                      this.EncryptRequired = ConvertStringToBool(headerInfo[index][1],
headerInfo[index][0], index);
//                  }

//                  if (headerInfo[index][0].ToUpper() == "INLINE ENABLED NTH DEVICE")
//                  {
//                      this.InlineNthDevice = ConvertStringToInt(headerInfo[index][1],
headerInfo[index][0], index);
//                  }

//                  if (headerInfo[index][0].ToUpper() == "LOG NTH DEVICE")
//                  {
//                      this.LogNthDevice = ConvertStringToInt(headerInfo[index][1],
headerInfo[index][0], index);
//                  }
```

```
//                    if (headerInfo[index][0].ToUpper() == "NUMBER OF SITES")
//                    {
//                        this.NumberOfsites = ConvertStringToInt(headerInfo[index][1],
headerInfo[index][0], index);
//                    }

//                }
//            }

//            // Notify the View-Model that the product config data has been read in.
//            OnPropertyChanged("CsvDataLoaded");
//        }
//        catch (Exception ex)
//        {
//            string message = string.Format("An error occured while loading a limit file {0}. {1}",
FileName, ex.Message);
//            TestStudioModel.trace.TraceInformation(message);
//            throw new ProductConfigurationException(message);
//        }
//    }

//    /// <summary>
//    /// Method that causes the data in the to be written to the file associated with this instance of
the Product Configuration viewModel.
//    /// </summary>
//    public void SaveDataFile(string FileName)
//    {
//        try
//        {
//            // Define some settings for the format of the XML file.
//            XmlWriterSettings xmlWriterSettings = new XmlWriterSettings();
//            xmlWriterSettings.NewLineOnAttributes = false;
//            xmlWriterSettings.Indent = true;

//            //System.Security.Cryptography.MACTripleDES hash = new
System.Security.Cryptography.MACTripleDES(Encoding.Default.GetBytes("mykey"));
//            //string hashString =
Convert.ToBase64String(hash.ComputeHash(Encoding.Default.GetBytes(myXMLString)));

//            using (XmlWriter xmlWriter = XmlWriter.Create(FileName, xmlWriterSettings))
//            {
```

```
//                xmlWriter.WriteStartDocument();
//                xmlWriter.WriteStartElement("ProductConfig",
"http://tempuri.org/XMLSchema1.xsd");
//                xmlWriter.WriteAttributeString("Version", "1");
//                //xmlWriter.WriteAttributeString("xmlns", "");

//                xmlWriter.WriteStartElement("Settings");


//                xmlWriter.WriteStartElement("ContinueOnFail");
//                xmlWriter.WriteValue(this.ContinueOnFail);
//                xmlWriter.WriteEndElement(); // Continue On Fail

//                xmlWriter.WriteStartElement("StopOnFail");
//                xmlWriter.WriteValue(this.StopOnFail);
//                xmlWriter.WriteEndElement(); // Continue On Fail

//                xmlWriter.WriteStartElement("ContinueOnAllFail");
//                xmlWriter.WriteValue(this.ContinueOnAllFail);
//                xmlWriter.WriteEndElement(); // Continue On Fail

//                xmlWriter.WriteStartElement("StopOnAlarm");
//                xmlWriter.WriteValue(this.StopOnalarm);
//                xmlWriter.WriteEndElement(); // Continue On Fail

//                xmlWriter.WriteStartElement("GoldUnitEnable");
//                xmlWriter.WriteValue(this.GoldUnitEnable);
//                xmlWriter.WriteEndElement(); // Continue On Fail

//                xmlWriter.WriteStartElement("UserCalibration");
//                xmlWriter.WriteValue(this.UserCalibrationEnable);
//                xmlWriter.WriteEndElement(); // Continue On Fail


//                xmlWriter.WriteStartElement("CalibrationExpiration");
//                xmlWriter.WriteValue(this.CalibrationExpiration);
//                xmlWriter.WriteEndElement(); // Continue On Fail

//                xmlWriter.WriteStartElement("OfflineQAEnable");
//                xmlWriter.WriteValue(this.OfflineQaAvailable);
//                xmlWriter.WriteEndElement(); // Continue On Fail
```

```
//                xmlWriter.WriteStartElement("EncryptRequired");
//                xmlWriter.WriteValue(this.EncryptRequired);
//                xmlWriter.WriteEndElement(); // Continue On Fail

//                xmlWriter.WriteStartElement("InlineEnabledNthDevice");
//                xmlWriter.WriteValue(this.InlineNthDevice);
//                xmlWriter.WriteEndElement(); // Continue On Fail

//                xmlWriter.WriteStartElement("LogNthDevice");
//                xmlWriter.WriteValue(this.LogNthDevice);
//                xmlWriter.WriteEndElement(); // Continue On Fail


//                xmlWriter.WriteStartElement("NumberOfSites");
//                xmlWriter.WriteValue(this.NumberOfsites);
//                xmlWriter.WriteEndElement(); // Continue On Fail


//                xmlWriter.WriteEndElement(); // Settings

//              //  xmlWriter.WriteEndElement(); // ProductConfig

//                xmlWriter.WriteEndDocument();
//            }

//            XmlTextReader r = new XmlTextReader(FileName);
//            while (r.Read())
//            {
//                if (r.NodeType == XmlNodeType.Element &&
//                r.Name == "Settings")
//                    Console.WriteLine(r.ReadOuterXml());
//            }
//            r.Close();

//            //using (StreamWriter sw = new StreamWriter(FileName))
//            //{
//            //    sw.WriteLine("Continue On Fail, {0}", this.ContinueOnFail.ToString().ToUpper());
//            //    sw.WriteLine("Stop On Fail, {0}", this.StopOnFail.ToString().ToUpper());
//            //    sw.WriteLine("Continue On All Fail, {0}",
this.ContinueOnAllFail.ToString().ToUpper());
//            //    sw.WriteLine("Stop On Alarm,{0}", this.StopOnalarm.ToString().ToUpper());
//            //    sw.WriteLine("Continue On Alarm, {0}",
```

```
this.ContinueOnAlarm.ToString().ToUpper());
//              //    sw.WriteLine("Gold Unit Enable / Disable,{0}",
this.GoldUnitEnable.ToString().ToUpper());
//              //    sw.WriteLine("User Calibration,{0}",
this.UserCalibrationEnable.ToString().ToUpper());
//              //    sw.WriteLine("Calibration Expiration,{0}", this.CalibrationExpiration.ToString());
//              //    sw.WriteLine("Offline QA Enable,{0}",
this.OfflineQaAvailable.ToString().ToUpper());
//              //    sw.WriteLine("Encrypt Required,{0}", this.EncryptRequired.ToString().ToUpper());
//              //    sw.WriteLine("Inline enabled nth device,{0}", this.InlineNthDevice);
//              //    sw.WriteLine("Log nth Device,{0}", this.LogNthDevice);
//              //    sw.WriteLine("Number of Sites,{0}", this.NumberOfsites);
//              //}
//          }
//          catch (Exception ex)
//          {
//              string message = string.Format("An error occured while saving a product configuration
file {0}. {1}", FileName, ex.Message);
//              TestStudioModel.trace.TraceInformation(message);
//              throw new ProductConfigurationException(message);
//          }
//      }

//      ///// <summary>
//      ///// Method that causes the data in the to be written to the file associated with this instance
of the Product Configuration viewModel.
//      ///// </summary>
//      //public void SaveCsvDataFile(string FileName)
//      //{
//      //    try
//      //    {
//      //        using (StreamWriter sw = new StreamWriter(FileName))
//      //        {
//      //            sw.WriteLine("Continue On Fail, {0}", this.ContinueOnFail.ToString().ToUpper());
//      //            sw.WriteLine("Stop On Fail, {0}", this.StopOnFail.ToString().ToUpper());
//      //            sw.WriteLine("Continue On All Fail, {0}",
this.ContinueOnAllFail.ToString().ToUpper());
//      //            sw.WriteLine("Stop On Alarm,{0}", this.StopOnalarm.ToString().ToUpper());
//      //            sw.WriteLine("Continue On Alarm, {0}",
this.ContinueOnAlarm.ToString().ToUpper());
//      //            sw.WriteLine("Gold Unit Enable / Disable,{0}",
this.GoldUnitEnable.ToString().ToUpper());
```

```
//      //          sw.WriteLine("User Calibration,{0}",
this.UserCalibrationEnable.ToString().ToUpper());
//      //          sw.WriteLine("Calibration Expiration,{0}", this.CalibrationExpiration.ToString());
//      //          sw.WriteLine("Offline QA Enable,{0}",
this.OfflineQaAvailable.ToString().ToUpper());
//      //          sw.WriteLine("Encrypt Required,{0}", this.EncryptRequired.ToString().ToUpper());
//      //          sw.WriteLine("Inline enabled nth device,{0}", this.InlineNthDevice);
//      //          sw.WriteLine("Log nth Device,{0}", this.LogNthDevice);
//      //          sw.WriteLine("Number of Sites,{0}", this.NumberOfsites);
//      //      }
//      //  }
//      //  catch (Exception ex)
//      //  {
//      //      string message = string.Format("An error occured while saving a product
configuration file {0}. {1}", FileName, ex.Message);
//      //      TestStudioModel.trace.TraceInformation(message);
//      //      throw new TestLimitException(message);
//      //  }
//      //}

//      /// <summary>
//      /// Create the OnPropertyChanged method to raise the event.
//      /// </summary>
//      /// <param name="name">Name of the property that's just been updated.</param>
//      public void OnPropertyChanged(string name)
//      {
//          PropertyChangedEventHandler handler = PropertyChanged;

//          if (handler != null)
//          {
//              handler(this, new PropertyChangedEventArgs(name));
//          }
//      }

//#endregion Public Methods
//  }
}
```

ProjectConfigModel.cs

```
using MT.TestStudio.Exceptions;
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace MerlinTest.Tools.TestStudio.Model
{
/// <summary>
/// Project Config Model class.
/// </summary>
public class ProjectConfigModel
{
#region Private Member Variables

#endregion Private Member Variables

#region Public Properties

public string UserTargetSystem { get; set; }

/// <summary>
/// Gets / Sets the project filename.
/// </summary>
public string ProjectFileName { get; set; }

/// <summary>
/// Gets / Sets the project file version..
/// </summary>
public int ProjectFileVersion { get; set; }

/// <summary>
/// Gets / Sets the project file path..
/// </summary>
public string ProjectFilePath { get; set; }

/// <summary>
/// Gets / Sets the project name.
/// </summary>
public string ProjectName { get; set; }
```

```
/// <summary>
/// Gets / Sets the project name.
/// </summary>
public string ProductName { get; set; }

/// <summary>
/// Gets / Sets the project date.
/// </summary>
public DateTime ProjectDate { get; set; }

/// <summary>
/// Gets / Sets the project comment.
/// </summary>
public string ProjectComment { get; set; }

#endregion Public Properties

#region Public Events / Delegates

/// <summary>
/// Event of type PropertyChangedEventHandler which is used to implement
INotifyPropertyChanged
/// and provide notification of when a property has changed.
/// </summary>
public event PropertyChangedEventHandler PropertyChanged;

#endregion Public Events / Delegates

#region Public Methods

/// <summary>
/// Helper Method to convert a string to an int.
/// </summary>
/// <param name="Value">Value to convert.</param>
/// <param name="ErrorText">Error text to display in exception.</param>
/// <returns>The converted value.</returns>
private static int ConvertStringToInt(string Value, string ErrorText)
{
if (int.TryParse(Value, out int result) == true)
{
return result;
}
```

```
else
{
throw new ProjectConfigurationException(string.Format("{0} - Value of {1} is invalid. ", ErrorText,
Value));
}
}

/// <summary>
/// Helper Method to convert a string to a date.
/// </summary>
/// <param name="Value">Value to convert.</param>
/// <param name="ErrorText">Error text to display in exception.</param>
/// <returns>The converted value.</returns>
private static DateTime ConvertStringToDateTime(string Value, string ErrorText)
{
if (DateTime.TryParse(Value, out DateTime result) == true)
{
return result;
}
else
{
throw new ProjectConfigurationException(string.Format("{0} - Value of {1} is invalid. ", ErrorText,
Value));
}
}

/// <summary>
/// Method that causes a file to be read in.
/// </summary>
/// <param name="FileName">The file to be read in.</param>
public void LoadCsvDataFile(string FileName)
{
try
{
XDocument document = XDocument.Load(FileName);

this.ProjectFileName = Path.GetFileName(FileName);

this.ProjectFilePath = Path.GetDirectoryName(FileName);

// Read Basic Project File Meta Data.
// Find all Files under the HWConfig XML node in the XML file.
```

```
this.ProjectName = (from myConfig in document.Elements("Project")
select myConfig.Attribute("Name").Value)
.First();

string version = (from myConfig in document.Elements("Project")
select myConfig.Attribute("Version").Value)
.First();

this.ProjectFileVersion = ConvertStringToInt(version, "Project File Version");

string dateTime = (from myConfig in document.Elements("Project")
select myConfig.Attribute("ProjectCreationDate").Value)
.First();

this.ProjectDate = ConvertStringToDateTime(dateTime, "Project Date");

this.ProductName = (from myConfig in document.Elements("Project")
select myConfig.Attribute("Product").Value)
.First();

this.ProjectComment = (from myConfig in document.Elements("Project")
select myConfig.Attribute("ProjectComment").Value)
.First();

this.UserTargetSystem = (from myConfig in document.Elements("Project")
select myConfig.Attribute("UserTargetSystem").Value)
.First();

// Notify the View-Model that the product config data has been read in.
OnPropertyChanged("CsvDataLoaded");
}
catch (Exception ex)
{
string message = string.Format("An error occured while loading a project configuration file {0}.
{1}", FileName, ex.Message);
TestStudioModel.trace.TraceInformation(message);
throw new TestLimitException(message);
}

}

/// <summary>
```

```
/// Method that causes the data in the to be written to the file associated with this instance of the
Product Configuration viewModel.
/// </summary>
public void SaveCsvDataFile(string FileName)
{
try
{
// Load Project XML file.
XDocument document = XDocument.Load(FileName);

var query = from c in document.Elements("Project")
select c;

foreach (XElement project in query)
{
project.Attribute("Name").Value = this.ProjectName;
project.Attribute("ProjectComment").Value = this.ProjectComment;
project.Attribute("UserTargetSystem").Value = this.UserTargetSystem;
}



// Save project file to reflect changes.
document.Save(FileName);
}
catch (Exception ex)
{
string message = string.Format("An error occured while saving a project setting file {0}. {1}",
FileName, ex.Message);
TestStudioModel.trace.TraceInformation(message);
throw new TestLimitException(message);
}
}

/// <summary>
/// Create the OnPropertyChanged method to raise the event.
/// </summary>
/// <param name="name">Name of the property that's just been updated.</param>
public void OnPropertyChanged(string name)
{
PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null)
```

```
{
handler(this, new PropertyChangedEventArgs(name));
}
}

#endregion Public Methods
}
}
```

TestStudioModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MerlinTest.Tools.Diagnostic.Logging;
using System.ComponentModel;
using System.Collections.ObjectModel;
using System.Threading;
using System.Threading.Tasks;
using System.Runtime.InteropServices;     // DLL support
using System.IO;
using System.Diagnostics;
using System.Net;
using System.Net.Sockets;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

namespace MerlinTest.Tools.TestStudio.Model
{
/// <summary>
/// Class that acts as the MODEL in the MVVM pattern. It implements the business logic of the application.
/// </summary>
public class TestStudioModel
{
#region Private

/// <summary>
/// Represents the status of the model object.
/// </summary>
private string status;
```

```
#endregion Private

#region Public Events / Delegates

/// <summary>
/// Event of type PropertyChangedEventHandler which is used to implement
INotifyPropertyChanged and provide notification of when a property has changed.
/// </summary>
public event PropertyChangedEventHandler PropertyChanged;

#endregion Public Events / Delegates

#region Constructor

/// <summary>
/// Constructor for the MODEL class.
/// </summary>
public TestStudioModel()
{
// Set the default status to ready.
status = "Ready";
}

#endregion Constructor

#region Public Properties

/// <summary>
/// Static custom trace object so that all models can log messages.
/// </summary>
public static CustomTraceSource trace = new CustomTraceSource("Model");

/// <summary>
/// Property to hold a message which represents the current status.
/// </summary>
public string Status
{
get
{
return status;
}
set
```

```
{
status = value;
// Now notify the viewModel that the StatusBar needs refreshing.
OnPropertyChanged("StatusBar");
}
}

#endregion Public Properties

#region Public Methods

/// <summary>
/// Create the OnPropertyChanged method to raise the event.
/// </summary>
/// <param name="name">Name of the property that's just been updated.</param>
public void OnPropertyChanged(string name)
{
PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null)
{
handler(this, new PropertyChangedEventArgs(name));
}
}

#endregion Public Methods
}
}
```

.NETFramework,Version=v4.6.1.AssemblyAttributes.cs

```
// <autogenerated />
using System;
using System.Reflection;
[assembly:
global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETFramework,Version=v4.6.1",
FrameworkDisplayName = ".NET Framework 4.6.1")]
```

AssemblyInfo.cs

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
```

```
// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Merlin Test Studio Model")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("Merlin Test Studio Model")]
[assembly: AssemblyCopyright("Copyright ©  2019")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("b29359f7-61e5-484c-958c-7a81cfb68127")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyFileVersion("0.14.0.01")]
```

DllFile.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Reflection;
using System.Runtime.InteropServices;
using System.Xml.Serialization;
```

```
namespace MerlinTestStudio.DataModels
{
[Serializable]
public class DllType
{
public Type SelfType { get; set; }
public List<MethodInfo> Methods { get; set; }
}

[Serializable]
public class DllFile
{
public string Version { get; set; }
public string Culture { get; set; }
public string PublicKeyToken { get; set; }
public string FileName { get; set; }
public DateTime LastModified { get; set; }
public string FileType { get; set; }
public string Description { get; set; }

public List<DllType> Types { get; set; }

#region Constructors
public DllFile() { } //Deserialization Constructor
public DllFile(string filePath)
{
FileInfo fileInfo = new FileInfo(filePath);
FileName = fileInfo.Name;
LastModified = fileInfo.LastWriteTime;
FileType = fileInfo.Extension;

Assembly assembly = Helpers.AssemblyLoader.LoadWithDependencies(filePath);
//Assembly.LoadFile(filePath);
Version = assembly.GetName().Version.ToString();
Culture = assembly.GetName().CultureName;
PublicKeyToken = BitConverter.ToString(assembly.GetName().GetPublicKeyToken()).Replace("-",
"");

Description = GetFileDescription(filePath);

Types = new List<DllType>();
foreach (Type type in assembly.GetTypes())
```

```
{
if (type.IsPublic == true && type.IsEnum != true)
{
//Gets and then sets the added functions.
var methodLibrary = new List<MethodInfo>();

foreach (MethodInfo MI in type.GetMethods(BindingFlags.Instance | BindingFlags.Public |
BindingFlags.DeclaredOnly))
{
if (MI.IsSpecialName != true && MI.IsVirtual != true)
{
methodLibrary.Add(MI);
}
}

Types.Add(new DllType()
{
SelfType = type,
Methods = methodLibrary
});
}
}
}
#endregion Constructors

private string GetFileDescription(string filePath)
{
var fileVersionInfo = FileVersionInfo.GetVersionInfo(filePath);
return fileVersionInfo.FileDescription;
}

#region Import / Save Methods

public void SaveToXml(string FilePath)
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(FilePath))
{
xmlSerializer.Serialize(writer, this);
}
```

```
}

public static DllFile LoadFromXml(string FilePath)
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(DllFile));

DllFile dllFileAfterDeSer;

// Deserialize XML file.
using (StreamReader sr = new StreamReader(FilePath))
{
dllFileAfterDeSer = (DllFile)ser.Deserialize(sr);
}

return dllFileAfterDeSer;
}

#endregion Import / Save Methods
}

}
```

Enums.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio.DataModels
{
//public enum GoldControl
//{
//    Do_Nothing,
//    Check_To_Baseline,
//    Check_To_Retest
//}
//public enum OffsetControl
//{
//    Do_Nothing,
```

```
//    Apply_To_Result,
//    Use_During_Test
//}
}
```

PinMapData.cs

```
using MerlinTest.Common.Types;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace MerlinTestStudio.DataModels
{
[Serializable]
public class Pin
{
private List<Mapping> _mappings = new List<Mapping>();

public string PinName { get; set; }
public string PinAlias { get; set; }
public List<Mapping> mappings { get { return _mappings; } set { _mappings = value; } }
}

[Serializable]
public class Mapping
{
public string InstrumentName { get; set; }
public string IO { get; set; }
public int Board { get; set; }
}

/// <summary>
/// Exportable data model to the new commonly used types Library.
/// </summary>
[Serializable]
[XmlRoot("PinMap")]
public class PinMapData
{
```

```
public string FilePath { get; set; }
public double Version { get; set; }

public List<Pin> RfPins { get; set; }
public List<Pin> DigitalPins { get; set; }
public List<Pin> PowerSupplyPins { get; set; }

#region Constructor
public PinMapData() { }
#endregion

#region Public Methods

/// <summary>
/// Returns a single mapping from digital pins using pin name and site enum as search parameters.
/// </summary>
/// <param name="pinName"></param>
/// <param name="site"></param>
/// <returns></returns>
public Mapping GetDigitalPinMap(string pinName, Site site)
{
Mapping rMapping = null;

//Gets the mappings of the specified pin.
var rPinMappingsLinq = DigitalPins.Where(x => x.PinName == pinName).Select(x =>
x.mappings).FirstOrDefault();

//Check if pin mappings are not null and if the index for the required site is within range.
if (rPinMappingsLinq != null && rPinMappingsLinq.Count() >= ((int)site + 1))
{
rMapping = rPinMappingsLinq[(int)site];//Select the mapping based on the site enum underlying
value (as an index).
rMapping.Board = GetBoardNumberFromInstrumentName(rMapping.InstrumentName); //Set
board number.
}

return rMapping;
}

/// <summary>
/// Returns the mappings of a single pin from digital pins using pin name as the search parameter.
/// </summary>
```

```
/// <param name="pinName"></param>
/// <returns></returns>
public List<Mapping> GetDigitalPinMap(string pinName)
{
//Gets the mappings of the specified pin
var rPinMappingsLinq = DigitalPins.Where(x => x.PinName == pinName).Select(x =>
x.mappings).FirstOrDefault();

//Board numbers postproccessing.
if (rPinMappingsLinq != null)
{
foreach (Mapping dpm in rPinMappingsLinq)
{
dpm.Board = GetBoardNumberFromInstrumentName(dpm.InstrumentName);
}
}

return rPinMappingsLinq;
}

//Temporary Helper Method.
public int GetBoardNumberFromInstrumentName(string instrumentName)
{
if (!string.IsNullOrEmpty(instrumentName)) //If it is not null or empty string, then proceed with
regex processing.
{
return int.Parse(System.Text.RegularExpressions.Regex.Match(instrumentName.Split(' ')[1],
@"\d+").Value);
}
else { return 0; }
}

#region Import / Save Methods

/// <summary>
/// Writes the current object to XML.
/// </summary>
/// <param name="Filename">The filename and path of the file to write.</param>
public void SaveToXml(string FilePath)
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());
```

```
// Serialize to disk.
using (StreamWriter writer = new StreamWriter(FilePath))
{
xmlSerializer.Serialize(writer, this);
}
}

/// <summary>
/// Imports a PIN Map file in the XML format.
/// </summary>
/// <param name="PinMapXmlFile">The filename and path of the XML based PIN Map
file.</param>
/// <returns>A calibration object.</returns>
public static PinMapData ImportPinMapXML(string FilePath)
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(PinMapData));

PinMapData pinMapAfterDeSerialize;

// Deserialize XML file.
using (StreamReader sr = new StreamReader(FilePath))
{
pinMapAfterDeSerialize = (PinMapData)ser.Deserialize(sr);
}

return pinMapAfterDeSerialize;
}

#endregion Import / Save Methods

#endregion Public Methods

}
}
```

ProductConfiguration.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace MerlinTestStudio.DataModels
{
public class ProductConfiguration
{
public bool ContinueOnFail { get; set; }
public bool StopOnFail { get; set; }
public bool ContinueOnAllFail { get; set; }
public bool StopOnalarm { get; set; }
public bool ContinueOnAlarm { get; set; }
public bool GoldUnitEnable { get; set; }
public bool UserCalibrationEnable { get; set; }
public int CalibrationExpiration { get; set; }
public bool OfflineQaAvailable { get; set; }
public bool EncryptRequired { get; set; }
public int InlineNthDevice { get; set; }
public int LogNthDevice { get; set; }
public int NumberOfsites { get; set; }


public ProductConfiguration()
{

}


#region Import / Save Methods

/// <summary>
/// Writes the current object to XML.
/// </summary>
/// <param name="Filename">The filename and path of the file to write.</param>
public void SaveToXml(string FilePath)
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(FilePath))
{
```

```
xmlSerializer.Serialize(writer, this);
}
}

/// <summary>
/// Imports a PIN Map file in the XML format.
/// </summary>
/// <param name="PinMapXmlFile">The filename and path of the XML based PIN Map
file.</param>
/// <returns>A calibration object.</returns>
public static ProductConfiguration LoadFromXml(string FilePath)
{
XmlSerializer ser = null;

ser = new XmlSerializer(typeof(ProductConfiguration));

ProductConfiguration productConfigurationAfterDeSer;

// Deserialize XML file.
using (StreamReader sr = new StreamReader(FilePath))
{
productConfigurationAfterDeSer = (ProductConfiguration)ser.Deserialize(sr);
}

return productConfigurationAfterDeSer;
}

#endregion Import / Save Methods



}
}
```

RffePattern.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;
```

```
namespace MerlinTestStudio.DataModels
{

[Serializable]
[XmlRoot("DigitalPatternModel")]
//[XmlRoot("RffePattern")]
public class RffePattern
{
private const string dpatExt = ".dpat";
private const string dpatsrcExt = ".dpatsrc";

public string FilePath { get; set; }

public string TimeSetUsed { get; set; }

public string ClockPinItem { get; set; }

public string DataPinItem { get; set; }

public List<string> Registers { get; set; }

public bool IsCondensed { get; set; }

public List<Mode> ModeCollection { get; set; }


#region XML Import/Export
public void SaveToXml(string FileName)
{
try
{
XmlSerializer xmlSerializer = new XmlSerializer(this.GetType());

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(FileName))
{
xmlSerializer.Serialize(writer, this);
}
}
catch (Exception ex) { throw ex; }
}
public static RffePattern LoadFromXml(string FileName)
```

```
{
try
{
XmlSerializer ser = new XmlSerializer(typeof(RffePattern));
RffePattern rffePatternAfterDeSerialize;

using (StreamReader sr = new StreamReader(FileName))// Deserialize XML file.
{
rffePatternAfterDeSerialize = (RffePattern)ser.Deserialize(sr);
}

return rffePatternAfterDeSerialize;
}
catch (Exception ex) { throw ex; }
}
#endregion

#region Methods
//NEEDS TESTING
public List<string> GetSubPatternFilePaths(string fileExtension)
{
List<string> paths = new List<string>();

switch (fileExtension)
{
case dpatExt:
case dpatsrcExt:
string dir = this.GetSubPatternsDirectory();
foreach (Mode subPattern in this.ModeCollection)
{
string fileName = subPattern.Name + fileExtension;
string fullFilePath = Path.Combine(dir, fileName);
paths.Add(fullFilePath);
}
return paths;

default: //Not an allowed file extension.
return paths; //Return Empty list.
}
}

//NEEDS TESTING
```

```
public List<string> GetSubPatternFileNames(string fileExtension)
{
List<string> FileNames = new List<string>();

switch (fileExtension)
{
case dpatExt:
case dpatsrcExt:
foreach (var subPattern in this.ModeCollection)
{
FileNames.Add(subPattern.Name + fileExtension);
}
return FileNames;

default: //Not an allowed file extension.
return FileNames; //Return Empty list.
}
}

private string GetSubPatternsDirectory()
{
string dirWithSubPatterns;

dirWithSubPatterns = Path.Combine(Path.GetDirectoryName(this.FilePath),
Path.GetFileNameWithoutExtension(this.FilePath));

return dirWithSubPatterns;
}
#endregion Methods
}

[Serializable]
public class Mode //Sub Pattern
{
public string Name { get; set; } //Sub Pattern name also used in file name.
public List<ModeVector> ModeVectors { get; set; } //List of mode data vectors with matching index
of it's corresponding register.
}

[Serializable]
public class ModeVector //Mode Data
{
```

```
public string InputValue { get; set; } //Data
public List<Operation> VectorizedData { get; set; } //List of dpatsrc vectors for file pattern compile.
}

[Serializable]
public enum VectorSection
{
Start_Sequence_Control,
Command,
Data,
Bus_Park
}

// Represents an operation in the pattern
[Serializable]
public class Operation //dpatsrc vector
{
public int VectorNumber { get; set; }

private string _opcode = "";
public string Opcode { get { return _opcode; } set { _opcode = value != null ? value : ""; } }

public string TimeSet { get; set; }
public string Clock { get; set; }
public string Data { get; set; }
public string Comment { get; set; }

[XmlIgnore]
public int SectionBitNumber => CalculateSectionBitNumber(this.VectorNumber);

[XmlIgnore]
public VectorSection? Section => CalculateSectionEnum(this.VectorNumber);

#region Constructors
public Operation() { } //Deserialization Constructor.
public Operation(string clock, string data)
{
Clock = clock;
Data = data;
}
public Operation(string name, string clock, string data)
{
```

```
TimeSet = name;
Clock = clock;
Data = data;
}
#endregion

#region Methods and Overrides

/// <summary>
/// Returns a boolean value that checks if this Operation instance is, not equal, but repeatable over the given Operation parameter.
/// </summary>
/// <param name="obj"></param>
/// <returns></returns>
public bool RepeatEquals(Operation obj)
{
Operation other = obj as Operation;
if (other == null)
{
return false;
}
return Opcode == other.Opcode && TimeSet == other.TimeSet && Clock == other.Clock && Data == other.Data;
}

/// <summary>
/// Returns the Operation as a padded string with the correct spacing.
/// </summary>
/// <returns></returns>
public override string ToString()
{
int space = 32 - Opcode.Length;

return Opcode + ">\t".PadLeft(space) + TimeSet + "\t" + Clock + "  " + Data + ";" + "  " + Comment;
}

//NEEDS TESTING.
public int CalculateSectionBitNumber(int vectorNumber)
{
int bitNumber = 0;

if (vectorNumber >= 1 && vectorNumber <= 11)  //Start control seqeunce // 1 - 11
```

```
{
bitNumber = 1;
}
else if (vectorNumber >= 12 && vectorNumber <= 24) //Command //12 - 24
{
bitNumber = vectorNumber - 11;
}
else if (vectorNumber >= 25 && vectorNumber <= 33) //Data //25 - 33
{
bitNumber = vectorNumber - 24;
}
else if (vectorNumber >= 34 && vectorNumber <= 36) //Bus Park //34 - 36
{
bitNumber = vectorNumber - 33;
}

return bitNumber;
}

//NEEDS TESTING.
public VectorSection? CalculateSectionEnum(int vectorNumber)
{
//Start control seqeunce // 1 - 11
if (vectorNumber >= 1 && vectorNumber <= 11)
{
return VectorSection.Start_Sequence_Control;
}
else if (vectorNumber >= 12 && vectorNumber <= 24) //Command //12 - 24
{
return VectorSection.Start_Sequence_Control;
}
else if (vectorNumber >= 25 && vectorNumber <= 33) //Data //25 - 33
{
return VectorSection.Start_Sequence_Control;
}
else if (vectorNumber >= 34 && vectorNumber <= 36) //Bus Park //34 - 36
{
return VectorSection.Start_Sequence_Control;
}

return null;
}
```

```
# endregion Methods and Overrides
}

}
```

TestLimitModels.cs

```
using MerlinTest.Common.Types;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

//namespace MerlinTestStudio.DataModels
//{
//    #region Test Limits Data Models
//    //Current file extension in use: ".limits"
[Serializable]
public class UpperLowerLimitsData
{
public List<UpperLowerLimit> Data { get; set; }

#region Import / Save
public void SaveToXml(string FilePath)
{
try
{
System.Xml.Serialization.XmlSerializer xmlSerializer = new
System.Xml.Serialization.XmlSerializer(this.GetType());

// Serialize to disk.
using (StreamWriter writer = new StreamWriter(FilePath))
{
xmlSerializer.Serialize(writer, this);
}
}
catch (Exception ex) { Console.WriteLine(ex.Message); }
}
public static UpperLowerLimitsData LoadFromXml(string FilePath)
```

```
{
try
{
System.Xml.Serialization.XmlSerializer ser = null;

ser = new System.Xml.Serialization.XmlSerializer(typeof(UpperLowerLimitsData));

UpperLowerLimitsData testLimitFile;

// Deserialize XML file.
using (StreamReader sr = new StreamReader(FilePath))
{
testLimitFile = (UpperLowerLimitsData)ser.Deserialize(sr);
}

return testLimitFile;
}
catch (Exception ex) { Console.WriteLine(ex.Message); return null; }
}
#endregion Import / Save
}

[Serializable]
public class UpperLowerLimit
{
#region Test Info Properties
public int ID { get; set; }
public int TestNumber { get; set; }
public string TestName { get; set; }
public string Units { get; set; }
#endregion Test Info Properties

#region Upper Lower Limit Properties
public double FTLower { get; set; }
public double FTUpper { get; set; }
public double QALower { get; set; }
public double QAUpper { get; set; }
#endregion Upper Lower Limit Properties

#region Binning Properties
public int HardBinNumber { get; set; }
public string HardBinName { get; set; }
```

```
public string HardBinPF { get; set; }
public int SoftBinNumber { get; set; }
public string SoftBinName { get; set; }
public string SoftBinPF { get; set; }
public ObservableCollection<MultiPassBin> MultiPassBins { get; set; }
public ObservableCollection<MultiFailBin> MultiFailBins { get; set; }

#endregion Binning Properties
}
```

AssemblyLoader.cs

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;

namespace MerlinTestStudio.DataModels.Helpers
{
/// <summary>
/// Helper class to assist with loading of an assembly & any unresolved dependencies.
/// </summary>
public static class AssemblyLoader
{
private static readonly ConcurrentDictionary<string, bool> AssemblyDirectories = new
ConcurrentDictionary<string, bool>();

static AssemblyLoader()
{
AssemblyDirectories[GetExecutingAssemblyDirectory()] = true;
AppDomain.CurrentDomain.AssemblyResolve += ResolveAssembly;
}

public static Assembly LoadWithDependencies(string assemblyPath)
{
AssemblyDirectories[Path.GetDirectoryName(assemblyPath)] = true;
return Assembly.LoadFile(assemblyPath);
}
```

```
private static Assembly ResolveAssembly(object sender, ResolveEventArgs args)
{
string dependentAssemblyName = args.Name.Split(',')[0] + ".dll";
List<string> directoriesToScan = AssemblyDirectories.Keys.ToList();

foreach (string directoryToScan in directoriesToScan)
{
string dependentAssemblyPath = Path.Combine(directoryToScan, dependentAssemblyName);
if (File.Exists(dependentAssemblyPath))
return LoadWithDependencies(dependentAssemblyPath);
}
return null;
}

private static string GetExecutingAssemblyDirectory()
{
string codeBase = Assembly.GetExecutingAssembly().CodeBase;
var uri = new UriBuilder(codeBase);
string path = Uri.UnescapeDataString(uri.Path);
return Path.GetDirectoryName(path);
}
}
}
```

.NETFramework,Version=v4.6.1.AssemblyAttributes.cs

```
// <autogenerated />
using System;
using System.Reflection;
[assembly:
global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETFramework,Version=v4.6.1",
FrameworkDisplayName = ".NET Framework 4.6.1")]
```

AssemblyInfo.cs

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("MerlinTestStudio.DataModels")]
[assembly: AssemblyDescription("")]
```

```
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("MerlinTestStudio.DataModels")]
[assembly: AssemblyCopyright("Copyright ©  2022")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("fdd09347-67e1-4a9b-a775-5aab27b5e61f")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyFileVersion("0.14.0.01")]
```

BaseUnitBaseExpr.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace UnitConversionLib
{
public class BaseUnitBaseExpr
{
#region Constructors

/// <summary>
/// Initializes a new instance of the <see cref="T:System.Object"/> class.
```

```
/// </summary>
public BaseUnitBaseExpr()
{
}


/// <summary>
/// Initializes a new instance of the <see cref="T:System.Object"/> class.
/// </summary>
public BaseUnitBaseExpr(string @base, double pow = 1)
{
_base = @base;
this.pow = pow;
}

#endregion

public string _base;
public double pow;

public override string ToString()
{
return string.Format("{0}^{1}", _base, pow);
}
}
}
```

BaseUnitDev.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace UnitConversionLib
{
public class BaseUnitDev
{
public List<BaseUnitBaseExpr> soorate = new List<BaseUnitBaseExpr>();
public List<BaseUnitBaseExpr> makhraj = new List<BaseUnitBaseExpr>();

#region Methods
public bool IsScalic()
```

```
{
this.Simplice();
return this.IsEmpty();
}

public bool IsEmpty()
{
return this.soorate.Count == 0 && this.makhraj.Count == 0;
}

public static bool Equals(BaseUnitDev left, BaseUnitDev right)
{
left.Simplice();
right.Simplice();

var oLeft = left.soorate.Select(i => i).ToList();
oLeft.AddRange(left.makhraj.Select(i => new BaseUnitBaseExpr(i._base, i.pow * -1)));

var oRight = right.soorate.Select(i => i).ToList();
oRight.AddRange(right.makhraj.Select(i => new BaseUnitBaseExpr(i._base, i.pow * -1)));

if(oLeft.Count!=oRight.Count)
return false;

foreach (var l in oLeft)
{
var r = oRight.Where(i => i._base == l._base).ToList();

if (r.Count > 1)
throw new UnitConversionLibException("Nabayad in ettefagh biofte");

if (r.Count == 0)
return false;

if (r.Count == 1)
if (l.pow != r.Single().pow)
return false;
}

foreach (var r in oRight)
{
var l = oLeft.Where(i => i._base == r._base).ToList();
```

```
if (l.Count > 1)
throw new UnitConversionLibException("Nabayad in ettefagh biofte");

if (l.Count == 0)
return false;

if (l.Count == 1)
if (r.pow != l.Single().pow)
return false;
}

return true;
}

public void ApplyPower(double pow)
{
foreach (var s in this.soorate)
s.pow *= pow;

foreach (var m in this.makhraj)
m.pow *= pow;
}

public void Simplice()
{
var all = this.soorate.Select(i => new Tuple<string, double>(i._base, i.pow)).ToList();
all.AddRange(this.makhraj.Select(i => new Tuple<string, double>(i._base, -i.pow)));

var dist = all.Select(i => i.Item1).Distinct().ToList();

var newSoorate = new List<BaseUnitBaseExpr>();
var newMakhraj = new List<BaseUnitBaseExpr>();

foreach(var nm in dist)
{
double totPow = all.Where(i => i.Item1 == nm).Select(i => i.Item2).Sum();

if (totPow == 0)
continue;

if (totPow > 0)
```

```
newSoorate.Add(new BaseUnitBaseExpr(nm, totPow));

if (totPow < 0)
newMakhraj.Add(new BaseUnitBaseExpr(nm, -totPow));
}

this.soorate.Clear();
this.soorate.AddRange(newSoorate);
this.makhraj.Clear();
this.makhraj.AddRange(newMakhraj);
}

public override string ToString()
{
this.Simplice();
var s = this.soorate.Select(i => i.pow == 1 ? i._base : string.Format("{0}^{1}", i._base,
i.pow)).ToList();
var m = this.makhraj.Select(i => i.pow == 1 ? i._base : string.Format("{0}^{1}", i._base,
i.pow)).ToList();

var stringS = string.Join(" . ", s.ToArray());
var stringM = string.Join(" . ", m.ToArray());

if (string.IsNullOrEmpty(stringS))
stringS = "1";

if (string.IsNullOrEmpty(stringM))
stringM = "1";

string format;

if (!ContainsAnyCharacter(stringS, "*/^") && !ContainsAnyCharacter(stringM, "*/^"))
format = stringM.Equals("1") ? "{0}" : "{0}/{1}";
else
format = stringM.Equals("1") ? "{0}" : "({0})/({1})";

var buf = string.Format(format, stringS, stringM);
return buf;

}
```

```
private static bool ContainsAnyCharacter(string taerget, string characters)
{
foreach (var chr in characters)
{
if (taerget.Contains(chr.ToString()))
return true;
}

return false;
}


#endregion
}
}
```

Consts.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace UnitConversionLib
{
public static class Consts
{
public static readonly double Yotta = 1e+24;
public static readonly double Zetta = 1e+21;
public static readonly double Exa = 1e+18;
public static readonly double Peta = 1e+15;
public static readonly double Terra = 1e+12;
public static readonly double Giga = 1e+9;
public static readonly double Mega = 1e+6;
public static readonly double Kilo = 1e+3;
public static readonly double Hecto = 1e+2;
public static readonly double Deca = 1e+1;



public static readonly double Deci = 1e-2;
public static readonly double Centi = 1e-2;
public static readonly double Milli = 1e-3;
public static readonly double Micro = 1e-6;
```

```
public static readonly double Nano = 1e-9;
public static readonly double Pico = 1e-12;
public static readonly double Femto = 1e-15;
public static readonly double Atto = 1e-18;
public static readonly double Zeppto = 1e-21;
public static readonly double Yocto = 1e-24;
}
}
```

CustomDic.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace UnitConversionLib
{


}
```

Measurable.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace UnitConversionLib
{
public struct Measurable : IFormattable
{
/// <summary>
/// Initializes a new instance of the <see cref="T:System.Object"/> class.
/// </summary>
public Measurable(Unit unit, double amount)
{
_unit = unit;
_amount = amount;
}

#region Field-Props

private Unit _unit;
```

```
private double _amount;

public Unit Unit
{
get { return _unit; }
}

public double Amount
{
get { return _amount; }
}

#endregion

#region Operators

public static Measurable operator *(Measurable left, Measurable right)
{
if (left.Unit.Scale != 1)
if (left.Unit.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

if (right.Unit.Scale != 1)
if (right.Unit.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

Measurable buf = new Measurable(left._unit*right.Unit, left.Amount*right.Amount);

return buf;
}

public static Measurable operator *(Measurable meas, double coeff)
{
if (meas.Unit.Scale != 1)
if (meas.Unit.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

Measurable buf = new Measurable(meas._unit, meas.Amount*coeff);

return buf;
}
```

```
public static Measurable operator ^(Measurable meas, double pow)
{
if (meas.Unit.Scale != 1)
if (meas.Unit.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

Measurable buf = new Measurable(meas._unit ^ pow, Math.Pow(meas.Amount, pow));

return buf;
}

public static Measurable operator *(double coeff, Measurable meas)
{
return meas*coeff;
}

public static Measurable operator /(Measurable left, Measurable right)
{
if (left.Unit.Scale != 1)
if (left.Unit.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

if (right.Unit.Scale != 1)
if (right.Unit.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

Measurable buf = new Measurable(left._unit/right.Unit, left.Amount/right.Amount);

return buf;
}

public static Measurable operator /(Measurable meas, double coeff)
{
if (meas.Unit.Scale != 1)
if (meas.Unit.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

Measurable buf = new Measurable(meas._unit, meas.Amount/coeff);

return buf;
}
```

```
public static Measurable operator /(double coeff, Measurable meas)
{
Measurable buf = new Measurable(1/meas._unit, coeff/meas.Amount);

return buf;
}

public static Measurable operator +(Measurable left, Measurable right)
{
if (left.Unit.Scale != 1)
if (left.Unit.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

if (right.Unit.Scale != 1)
if (right.Unit.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

double coef;

if (!left.Unit.IsConvertibleTo(right._unit, out coef))
throw UnitConversionLibException.InconsistentUnits;

var leftAmou = left._amount;

var rightAmount = right._amount*coef;

Measurable buf = new Measurable(left._unit, leftAmou + rightAmount);

return buf;
}

public static bool operator ==(Measurable left, Measurable right)
{
return left.Equals(right);
}

public static bool operator !=(Measurable left, Measurable right)
{
return !left.Equals(right);
}
```

```
#endregion

#region Public Methods

public bool IsConvertibleTo(Unit unt)
{
double coef;
return this._unit.IsConvertibleTo(unt, out coef);
}

public Measurable ConvertTo(Unit unt)
{
double coef;

if (!this._unit.IsConvertibleTo(unt, out coef))
throw new InvalidOperationException(string.Format("Unit '{0}' is not convertible to '{1}'",
this._unit.LocalCaption, unt.LocalCaption));

return new Measurable(unt, this._amount/coef);
}

#endregion

#region Static

public static Measurable Parse(string meaus)
{
var pat = @"^((\d+\.?\d*)([Ee][+-]?(\d+))?)\s+(.*?)$";

if (!System.Text.RegularExpressions.Regex.IsMatch(meaus, pat))
throw new UnitConversionLibException(string.Format("Fomat mismatch:\r\n{0}", meaus));

var mt = System.Text.RegularExpressions.Regex.Match(meaus, pat);

var allVals = mt.Groups.Cast<System.Text.RegularExpressions.Group>().Select(i =>
i.Value).ToList();

var amount = mt.Groups[1].Value;

var unit = mt.Groups[5].Value;

var amountDbl = double.Parse(amount);
```

```
var untU = Unit.Parse(unit);

return new Measurable(untU, amountDbl);
}

public static bool TryParse(string meaus,out Measurable val)
{
try
{
val = Parse(meaus);
return true;
}
catch (Exception)
{
val = new Measurable();
return false;
}
}

public static implicit operator Measurable(string val)
{
var temp = Measurable.Parse(val);
// code to convert from int to SampleClass...

return temp;
}

public static implicit operator double(Measurable val)
{
if (!val._unit.NotIsScalic) //then is scalic
return val._amount*val._unit.GetTotalScales();

throw new Exception("Measurable cannot convert directly to double type.");
// code to convert from int to SampleClass...
}

#endregion


public override string ToString()
{
```

```
return this.ToString(null,null);
}


public string ToString(string format, IFormatProvider formatProvider)
{

if(string.IsNullOrEmpty(format))
return string.Format(formatProvider, "{0} {1}", this.Amount, this.Unit);
else
return string.Format(formatProvider, "{0} {1}", this.Amount.ToString(format), this.Unit);

}
}
}
```

ParsingClasses.cs

```
using System;
using System.Collections;
using System.Text.RegularExpressions;

namespace UnitConversionLib.UnitParse
{
public enum TokenType
{
None,
Number,
//Constant,
//Plus,
//Minus,
Multiply,
Divide,
Exponent,
//UnaryMinus,
//Sine,
//Cosine,
//Tangent,
LeftParenthesis,
RightParenthesis,
Unit
}
```

```
public struct ReversePolishNotationToken
{
public string TokenValue;
public TokenType TokenValueType;

public override string ToString()
{
return string.Format("{0} : {1}", TokenValue, TokenValueType);
}
}

public class ReversePolishNotation
{
private Queue output;
private Stack ops;

private string sOriginalExpression;

public string OriginalExpression
{
get { return sOriginalExpression; }
}

private string sTransitionExpression;

public string TransitionExpression
{
get { return sTransitionExpression; }
}

private string sPostfixExpression;

public string PostfixExpression
{
get { return sPostfixExpression; }
}

public ReversePolishNotation()
{
sOriginalExpression = string.Empty;
sTransitionExpression = string.Empty;
sPostfixExpression = string.Empty;
```

```
}

public void Parse(string Expression)
{
output = new Queue();
ops = new Stack();

sOriginalExpression = Expression;

string sBuffer = Expression.Replace(" ", ""); //.ToLower();
// captures numbers. Anything like 11 or 22.34 is captured
sBuffer = Regex.Replace(sBuffer, @"(?<number>\d+(\.\d+)?)", " ${number} ");
// captures these symbols: + - * / ^ ( )
sBuffer = Regex.Replace(sBuffer, @"(?<ops>[+\-*/^()])", " ${ops} ");
// captures alphabets. Currently captures the two math constants PI and E,
// and the 3 basic trigonometry functions, sine, cosine and tangent
sBuffer = Regex.Replace(sBuffer, "(?<alpha>(pi))", " ${alpha} ");
// trims up consecutive spaces and replace it with just one space
sBuffer = Regex.Replace(sBuffer, @"\s+", " ").Trim();

// The following chunk captures unary minus operations.
// 1) We replace every minus sign with the string "MINUS".
// 2) Then if we find a "MINUS" with a number or constant in front,
//    then it's a normal minus operation.
// 3) Otherwise, it's a unary minus operation.

// Step 1.
sBuffer = Regex.Replace(sBuffer, "-", "MINUS");
// Step 2. Looking for pi or e or generic number \d+(\.\d+)?
sBuffer = Regex.Replace(sBuffer, @"(?<number>(pi|(\d+(\.\d+)?)))\s+MINUS", "${number} -");
// Step 3. Use the tilde ~ as the unary minus operator
sBuffer = Regex.Replace(sBuffer, "MINUS", "~");

sTransitionExpression = sBuffer;

// tokenise it!
string[] saParsed = sBuffer.Split(" ".ToCharArray());
int i = 0;
double tokenvalue;
ReversePolishNotationToken token, opstoken;
for (i = 0; i < saParsed.Length; ++i)
{
```

```
token = new ReversePolishNotationToken();
token.TokenValue = saParsed[i];
token.TokenValueType = TokenType.None;


bool isDoubleOrUnit = Regex.IsMatch(saParsed[i], @"\d+\.?\d*") ||
Unit.ArealyRegistered(saParsed[i]);

if (isDoubleOrUnit)
{
if (double.TryParse(saParsed[i], out tokenvalue))
{
token.TokenValueType = TokenType.Number;
// If the token is a number, then add it to the output queue.
output.Enqueue(token);
}
else
{
var unt = Unit.ParseDirect(saParsed[i]);
token.TokenValueType = TokenType.Unit;
output.Enqueue(token);
}
}
else
{
var nm = saParsed[i];
switch (saParsed[i])
{
case "*":
token.TokenValueType = TokenType.Multiply;
if (ops.Count > 0)
{
opstoken = (ReversePolishNotationToken) ops.Peek();
// while there is an operator, o2, at the top of the stack
while (IsOperatorToken(opstoken.TokenValueType))
{
// Once we're in here, the following algorithm condition is satisfied.
// o1 is associative or left-associative and its precedence is less than (lower precedence) or equal to that of o2, or
// o1 is right-associative and its precedence is less than (lower precedence) that of o2,

// pop o2 off the stack, onto the output queue;
```

```
output.Enqueue(ops.Pop());
if (ops.Count > 0)
{
opstoken = (ReversePolishNotationToken) ops.Peek();
}
else
{
break;
}
}
}
// push o1 onto the operator stack.
ops.Push(token);
break;
case "/":
token.TokenValueType = TokenType.Divide;
if (ops.Count > 0)
{
opstoken = (ReversePolishNotationToken) ops.Peek();
// while there is an operator, o2, at the top of the stack
while (IsOperatorToken(opstoken.TokenValueType))
{
// Once we're in here, the following algorithm condition is satisfied.
// o1 is associative or left-associative and its precedence is less than (lower precedence) or equal to that of o2, or
// o1 is right-associative and its precedence is less than (lower precedence) that of o2,

// pop o2 off the stack, onto the output queue;
output.Enqueue(ops.Pop());
if (ops.Count > 0)
{
opstoken = (ReversePolishNotationToken) ops.Peek();
}
else
{
break;
}
}
}
// push o1 onto the operator stack.
ops.Push(token);
break;
```

```
case "^":
token.TokenValueType = TokenType.Exponent;
// push o1 onto the operator stack.
ops.Push(token);
break;
case "(":
token.TokenValueType = TokenType.LeftParenthesis;
// If the token is a left parenthesis, then push it onto the stack.
ops.Push(token);
break;
case ")":
token.TokenValueType = TokenType.RightParenthesis;
if (ops.Count > 0)
{
opstoken = (ReversePolishNotationToken) ops.Peek();
// Until the token at the top of the stack is a left parenthesis
while (opstoken.TokenValueType != TokenType.LeftParenthesis)
{
// pop operators off the stack onto the output queue
output.Enqueue(ops.Pop());
if (ops.Count > 0)
{
opstoken = (ReversePolishNotationToken) ops.Peek();
}
else
{
// If the stack runs out without finding a left parenthesis,
// then there are mismatched parentheses.
throw new Exception("Unbalanced parenthesis!");
}
}
// Pop the left parenthesis from the stack, but not onto the output queue.
ops.Pop();
}

if (ops.Count > 0)
{
opstoken = (ReversePolishNotationToken) ops.Peek();
// If the token at the top of the stack is a function token
if (IsFunctionToken(opstoken.TokenValueType))
{
// pop it and onto the output queue.
```

```
output.Enqueue(ops.Pop());
}
}
break;
default:
{
var strtr = Expression.IndexOf(saParsed[i]) + 1;
var exc = new UnitParsingException("Error occurred during unit parsing."); //Original Error
Message:  string.Format("the phraze '{0}' not known at {1}'th character", saParsed[i], strtr)

exc.StartCharacter = strtr;
exc.UnknownPhraze = saParsed[i];

throw exc;
}
}
}
}

// When there are no more tokens to read:

// While there are still operator tokens in the stack:
while (ops.Count != 0)
{
opstoken = (ReversePolishNotationToken) ops.Pop();
// If the operator token on the top of the stack is a parenthesis
if (opstoken.TokenValueType == TokenType.LeftParenthesis)
{
// then there are mismatched parenthesis.
throw new UnitConversionLibException("Unbalanced parenthesis!");
}
else
{
// Pop the operator onto the output queue.
output.Enqueue(opstoken);
}
}

sPostfixExpression = string.Empty;
foreach (object obj in output)
{
opstoken = (ReversePolishNotationToken) obj;
```

```
sPostfixExpression += string.Format("{0} ", opstoken.TokenValue);
}
}

public Unit Evaluate()
{
Stack result = new Stack();
object oper1 = null, oper2 = null;

ReversePolishNotationToken token = new ReversePolishNotationToken();
// While there are input tokens left
foreach (object obj in output)
{
// Read the next token from input.
token = (ReversePolishNotationToken) obj;
switch (token.TokenValueType)
{
case TokenType.Number:
// If the token is a value
// Push it onto the stack.
result.Push(double.Parse(token.TokenValue));
break;
case TokenType.Unit:
// If the token is a value
// Push it onto the stack.
result.Push(Unit.ParseDirect(token.TokenValue));
break;
case TokenType.Multiply:
// NOTE: n is 2 in this case
// If there are fewer than n values on the stack
if (result.Count >= 2)
{
// So, pop the top n values from the stack.
oper2 = result.Pop();
oper1 = result.Pop();

if (oper1 is Unit)
{
if (oper2 is Unit)
{
result.Push((Unit) oper1*(Unit) oper2);
}
```

```
if (oper2 is double)
{
result.Push((Unit) oper1*(double) oper2);
}
}

if (oper1 is double)
{
if (oper2 is Unit)
{
result.Push((double) oper1*(Unit) oper2);
}
if (oper2 is double)
{
result.Push((double) oper1*(double) oper2);
}
}
// Evaluate the function, with the values as arguments.
// Push the returned results, if any, back onto the stack.
//result.Push(oper1 * oper2);
}
else
{
// (Error) The user has not input sufficient values in the expression.
throw new Exception("Evaluation error!");
}
break;
case TokenType.Divide:
// NOTE: n is 2 in this case
// If there are fewer than n values on the stack
if (result.Count >= 2)
{
oper2 = result.Pop();
oper1 = result.Pop();

if (oper1 is Unit)
{
if (oper2 is Unit)
{
result.Push((Unit) oper1/(Unit) oper2);
}
if (oper2 is double)
```

```
{
result.Push((Unit) oper1/(double) oper2);
}
}

if (oper1 is double)
{
if (oper2 is Unit)
{
result.Push((double) oper1/(Unit) oper2);
}
if (oper2 is double)
{
result.Push((double) oper1/(double) oper2);
}
}
}
else
{
// (Error) The user has not input sufficient values in the expression.
throw new Exception("Evaluation error!");
}
break;
case TokenType.Exponent:
// NOTE: n is 2 in this case
// If there are fewer than n values on the stack
if (result.Count >= 2)
{
oper2 = result.Pop();
oper1 = result.Pop();

if (oper1 is Unit)
{
/*if (oper2 is Unit)
{
throw new UnitConversionLibException("Unit Cannot place in power");
}
*/

if (oper2 is Unit)
if (((Unit) oper2).NotIsScalic)
throw new UnitConversionLibException("Unit Cannot place in power");
```

```
if (oper2 is double )
{
result.Push((Unit)oper1 ^ (double)oper2);
}

if ( oper2 is Unit)
{
result.Push((Unit)oper1 ^ ((Unit)oper2).GetTotalScales());
}
}
if (oper1 is double)
{
if (oper2 is Unit)
if (((Unit) oper2).NotIsScalic)
throw new UnitConversionLibException("Unit Cannot place in power");

if (oper2 is Unit)
{
result.Push(Math.Pow((double)oper1, ((Unit)oper2).GetTotalScales()));
}

if (oper2 is double)
{
result.Push(Math.Pow((double) oper1, (double) oper2));
}
}
}
else
{
// (Error) The user has not input sufficient values in the expression.
throw new Exception("Evaluation error!");
}
break;
}
}

// If there is only one value in the stack
if (result.Count == 1)
{
// That value is the result of the calculation.
var res = result.Pop();
```

```
return (Unit) res;
}
else
{
// If there are more values in the stack
// (Error) The user input too many values.
throw new Exception("Evaluation error!");
}
}

private bool IsOperatorToken(TokenType t)
{
bool result = false;
switch (t)
{
case TokenType.Multiply:
case TokenType.Divide:
case TokenType.Exponent:
result = true;
break;
default:
result = false;
break;
}
return result;
}

private bool IsFunctionToken(TokenType t)
{
bool result = false;
switch (t)
{
default:
result = false;
break;
}
return result;
}

private double EvaluateConstant(string TokenValue)
{
double result = 0.0;
```

```
switch (TokenValue)
{
case "pi":
result = Math.PI;
break;
case "e":
result = Math.E;
break;
}
return result;
}
}
}
```

TimerHelper.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;

namespace UnitConversionLib
{
public static class TimerHelper
{
public static string CreateNew(string st= null)
{
if (st == null)
st = Guid.NewGuid().ToString();

dic[st] = DateTime.Now;
return st;
}

public static TimeSpan GetDuration(string st)
{
return DateTime.Now - dic[st];
}
private static Dictionary<string, DateTime> dic = new Dictionary<string, DateTime>();



}
```

}

Unit.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Collections;
using UnitConversionLib.UnitParse;

namespace UnitConversionLib
{
public struct Unit : IEquatable<Unit>
{
#region Constructors

static Unit()
{
TimerHelper.CreateNew("as");



#region Base SI Units

var m = CreateNew("m"); //meter - length
abbreviationDic["M"] = "m";

var sec = CreateNew("s"); //second - time
abbreviationDic["sec"] = "s";
abbreviationDic["Sec"] = "s";

var kg = CreateNew("kg"); //kilograms - mass
abbreviationDic["Kg"] = "kg";

var a = CreateNew("A"); //Ampere - electric current
abbreviationDic["ampere"] = "A";
abbreviationDic["Ampere"] = "A";

var c = CreateNew("°C"); //centigrad degree - temprature
abbreviationDic["Celsius"] = "°C";
abbreviationDic["celsius"] = "°C";

var mol = CreateNew("mol"); //mol - amount of subtance
abbreviationDic["Mol"] = "mol";
```

```
abbreviationDic["MOL"] = "mol";

var cd = CreateNew("cd"); //candella - luminous intensity
abbreviationDic["Candela"] = "cd";
abbreviationDic["candela"] = "cd";
#endregion

#region Length

var angstrom = 1e-10 * m; //angstrom
Register(ref angstrom, "Å", false);
abbreviationDic["angstrom"] = "Å";
abbreviationDic["Angstrom"] = "Å";

var ft = 0.304800610 * m; //foot
Register(ref ft, "foot", false);
//Register(ref ft, "");

var inch = 0.0254 * m; //inch
Register(ref inch, "in", false);
abbreviationDic["inch"] = "in";
abbreviationDic["Inch"] = "in";

var micron = 1e-6 * m; //micron
Register(ref micron, "µ", false);
abbreviationDic["micron"] = "µ";

var mi = 1609.344 * m; //international mile
Register(ref mi, "mi", false);
abbreviationDic["Mile"] = "mi";
abbreviationDic["mile"] = "mi";

var yard = 0.9144 * m; //yard
Register(ref yard, "yd", false);
abbreviationDic["Yard"] = "yd";
abbreviationDic["yard"] = "yd";

var centimeter = 0.01 * m; //centimeter
Register(ref centimeter, "cm", false);
abbreviationDic["Centimeter"] = "cm";
abbreviationDic["centimeter"] = "cm";
```

```
var milimeter = 0.001 * m; //milimeter
Register(ref milimeter, "mm", false);
abbreviationDic["Milimeter"] = "cm";
abbreviationDic["milimeter"] = "cm";

var km = 1000 * m; //kilometer
Register(ref km, "km", false);
abbreviationDic["kilometer"] = "km";
abbreviationDic["Kilometer"] = "km";

#endregion



#region Time

var microsecond = 1e-6*sec; //microsecond
Register(ref microsecond, "µs", false);
abbreviationDic["microsecond"] = "µs";
abbreviationDic["Microsecond"] = "µs";


var milisecond = 1e-3*sec; //millisecond
Register(ref milisecond, "ms", false);
abbreviationDic["millisecond"] = "ms";
abbreviationDic["Millisecond"] = "ms";

var hour = 3600*sec; //hour
Register(ref hour, "h", false);
abbreviationDic["hour"] = "h";
abbreviationDic["Hour"] = "h";

var minute = 60*sec; //minute
Register(ref minute, "min", false);
abbreviationDic["Minute"] = "min";
abbreviationDic["minute"] = "min";

var day = 86400*sec; //day
Register(ref day, "d", false);
abbreviationDic["Day"] = "d";
abbreviationDic["day"] = "d";
```

```
var week = 7*86400*sec; //Week
Register(ref week, "week", false);
abbreviationDic["Week"] = "week";

var year = 365.24219878125*86400*sec; //year
Register(ref year, "yr", false);
abbreviationDic["year"] = "yr";
abbreviationDic["Year"] = "yr";

#endregion

#region Mass

var gr = 1e-3*kg; //grams
Register(ref gr, "gm", false);
abbreviationDic["gram"] = "gm";
abbreviationDic["Gram"] = "gm";

var mgr = 1e-6*kg; //mili grams
Register(ref mgr, "mg", false);
abbreviationDic["milligram"] = "mg";
abbreviationDic["Milligram"] = "mg";

var amu = 1.660538921e-27*kg; //atomic mass unit
Register(ref amu, "u", false);
abbreviationDic["Atomic mass unit"] = "u";
abbreviationDic["atomic mass unit"] = "u";

var slug = 14.5939*kg; //slug
Register(ref slug, "slug", false);
abbreviationDic["Slug"] = "slug";
abbreviationDic["pond"] = "slug";

var ton = 1e3 * kg; //ton
Register(ref ton, "ton", false);
abbreviationDic["Ton"] = "ton";

#endregion

#region Force

var N = kg*m/(sec ^ 2); //newton
```

```
Register(ref N, "N",false);

var dyne = 1e-5*N; //Dyne
Register(ref dyne, "dyn", false);
abbreviationDic["Dyne"] = "dyn";
abbreviationDic["dyne"] = "dyn";

var lbf = 4.4482216152605*N; //pound FOrce
Register(ref lbf, "lbf", false);

var kgf = 1e-1*N; //killogram-force
Register(ref kgf, "kgf", false);
abbreviationDic["Kgf"] = "kgf";

var tonForce = 1e2*N; //Tone FOrce
Register(ref tonForce, "tonf", false);
abbreviationDic["Tonf"] = "tonf";

#endregion

#region Pressure

var pas = N / (m ^ 2); //Pascal
Register(ref pas, "Pa",false);
abbreviationDic["pas"] = "Pa";
abbreviationDic["Pas"] = "Pa";

var kpas = 1e3 * pas; //Killo Pascal
Register(ref kpas, "KPa", false);
abbreviationDic["kpa"] = "KPa";
abbreviationDic["kpas"] = "KPa";
abbreviationDic["KPas"] = "KPa";

var mpas = 1e6 * pas; //mega Pascal
Register(ref mpas, "MPa", false);
abbreviationDic["mpa"] = "MPa";
abbreviationDic["MPas"] = "MPa";

var gpas = 1e9 * pas; //giga Pascal
Register(ref gpas, "GPa", false);
abbreviationDic["gpa"] = "GPa";
abbreviationDic["GPas"] = "GPa";
```

```
var bar = 1e5*pas; //Bar
Register(ref bar, "bar", false);
abbreviationDic["Bar"] = "bar";

var tAtm = 0.980665e5*pas; //Technical atmosphere
Register(ref tAtm, "at", false);
abbreviationDic["At"] = "at";

var atm = 1.01325e5*pas; //Atmosphere
Register(ref atm, "atm", false);
abbreviationDic["Atm"] = "atm";

var psi = 6.895*1e+3*pas; //Pound-force per square inch
Register(ref psi, "psi", false);
abbreviationDic["Psi"] = "psi";

var torr = 133.322*pas; //Pound-force per square inch
Register(ref torr, "torr", false);
abbreviationDic["Torr"] = "torr";

var mmH2O = 9.80665*pas; //milimeter water
Register(ref mmH2O, "mmH2O", false);
abbreviationDic["mmH2o"] = "mmH2O";

var mmHg = 133.322368421*pas; //milimeter Hg
Register(ref mmHg, "mmHg", false);

#endregion

var durr = TimerHelper.GetDuration("as");

}

public Unit(string name)
: this()
{
IsBase = true;
Name = name;
OverridedLocalName = name;
Scale = 1;
this.dev = new UnitDev();
```

```
this.NotIsScalic = true;
}

public Unit(UnitDev dev, double scale = 1)
: this()
{
this.dev = dev;

if (scale < 0)
throw new UnitConversionLibException("Scale Cannot Be Lower Than Zero");
Scale = scale;

this.NotIsScalic = !dev.GetOrigins().IsScalic();
}

#endregion

#region Methods

#region NonStatic

public BaseUnitDev GetLocalCaption()
{
if (Scale != 1)
{
if (OverridedLocalName == null)
throw new UnitConversionLibException("Unregistered Scalic Unit");
else
{
var tbuf = new BaseUnitDev();
tbuf.soorate.Add(new BaseUnitBaseExpr(OverridedLocalName));
return tbuf;
}
}

if (OverridedLocalName != null)
{
var tbuf = new BaseUnitDev();
tbuf.soorate.Add(new BaseUnitBaseExpr(OverridedLocalName));
return tbuf;
}
```

```
var s = this.dev.soorate.Select(i =>
{
var t = i._base.GetLocalCaption();
t.ApplyPower(i.pow);
return t;
});

var m = this.dev.makhraj.Select(i =>
{
var t = i._base.GetLocalCaption();
t.ApplyPower(i.pow);
return t;
});

var ss = s.SelectMany(i => i.soorate).ToList();
ss.AddRange(m.SelectMany(i => i.makhraj));

var mm = s.SelectMany(i => i.makhraj).ToList();
mm.AddRange(m.SelectMany(i => i.soorate));

var buf = new BaseUnitDev();

buf.soorate.AddRange(ss);
buf.makhraj.AddRange(mm);

buf.Simplice();

return buf;
}

internal BaseUnitDev GetOrigins()
{
if (this.IsBase)
{
var buf = new BaseUnitDev();
buf.soorate.Add(new BaseUnitBaseExpr(Name));
return buf;
}

if (!NotIsScalic)
return new BaseUnitDev();
```

```
return dev.GetOrigins();
}

internal double GetTotalScales()
{
if (this.IsBase)
return 1;

if (this.Scale == 0)
throw new UnitConversionLibException("Direct Created Unit");

var buf = this.Scale;

foreach (var r in this.dev.soorate)
buf *= Math.Pow(r._base.GetTotalScales(), r.pow);

foreach (var r in this.dev.makhraj)
buf /= Math.Pow(r._base.GetTotalScales(), r.pow);

return buf;
}

public Unit Copy()
{
var buf = new Unit();

buf.dev = this.dev.Copy();
buf.Scale = this.Scale;
buf.Name = this.Name;
buf.OverridedLocalName = this.OverridedLocalName;
buf.IsBase = this.IsBase;
buf.NotIsScalic = this.NotIsScalic;
return buf;
}

public bool IsConvertibleTo(Unit destUnit, out double scale)
{
scale = 0;

var thisOrg = this.GetOrigins();
var thatOrg = destUnit.GetOrigins();
if (BaseUnitDev.Equals(thisOrg, thatOrg))
```

```
{
var t = this.GetTotalScales();
var th = destUnit.GetTotalScales();
scale = th/t;
return true;
}

return false;
}

public override string ToString()
{
return string.Format("{0}", LocalCaption);
}

#region IEquatable

/// <summary>
/// Indicates whether the current object is equal to another object of the same type.
/// </summary>
/// <returns>
/// true if the current object is equal to the <paramref name="other"/> parameter; otherwise, false.
/// </returns>
/// <param name="other">An object to compare with this object.</param>
public bool Equals(Unit other)
{
double c;
if (this.IsConvertibleTo(other, out c))
if (c == 1)
return true;
return false;
}

/// <summary>
/// Indicates whether this instance and a specified object are equal.
/// </summary>
/// <returns>
/// true if <paramref name="obj"/> and this instance are the same type and represent the same
value; otherwise, false.
/// </returns>
/// <param name="obj">Another object to compare to. </param><filterpriority>2</filterpriority>
public override bool Equals(object obj)
```

```
{
if (ReferenceEquals(null, obj)) return false;
if (obj.GetType() != typeof (Unit)) return false;
return Equals((Unit) obj);
}

/// <summary>
/// Returns the hash code for this instance.
/// </summary>
/// <returns>
/// A 32-bit signed integer that is the hash code for this instance.
/// </returns>
/// <filterpriority>2</filterpriority>
public override int GetHashCode()
{
return 0;
}

#endregion

#endregion

#region Static



public static string GetGlobalName(Unit unt)
{
if (ArealyRegistered(unt))
return registrationDic.Where(i => i.Value.Equals(unt)).Single().Key;
throw new UnitConversionLibException(
string.Format("No unit with local caption '{0}' registered yet.", unt.LocalCaption));
}

private static void Register(ref Unit unt, string nm ,bool checkNotRegitered = true)
{
if (checkNotRegitered)
{

if (ArealyRegistered(nm))
throw new UnitConversionLibException(string.Format("unit with name '{0}' arealy registered.",
nm));
```

```
if (ArealyRegistered(unt))
throw new UnitConversionLibException(
string.Format("unit arealy registered."));

}

unt.OverridedLocalName = nm;
registrationDic[nm] = unt.Copy();
}

public static void Register(ref Unit unt, string nm)
{
Register(ref unt, nm, true);
}

public static Unit CreateNew(string nm)
{
if (ArealyRegistered(nm))
throw new UnitConversionLibException(string.Format("unit with name '{0}' arealy registered.",
nm));

var buf = new Unit(nm);
Register(ref buf, nm);
return buf;
}

public static Unit GetRegisteredUnit(string nm)
{
if (ArealyRegistered(nm))
if (abbreviationDic.Keys.Contains(nm))
return registrationDic[abbreviationDic[nm]];
else
return registrationDic[nm];
else
throw new UnitConversionLibException(string.Format("No unit with name '{0}' registered yet",
nm));
}

/// <summary>
/// Gets the appropriate unit tags with nm.
/// </summary>
```

```
/// <param name="nm">The nm.</param>
/// <returns></returns>
public static List<string> GetAppropriate(string nm)
{
var buf = new List<string>();


buf.AddRange(abbreviationDic.Values);

buf = buf.Distinct().ToList();

var wanted = buf.Where(i => i.ToLower() == nm.ToLower()).ToList();


if (wanted.Count == 0)
wanted = abbreviationDic.Keys.Where(i => i.ToLower() == nm.ToLower()).ToList();

return wanted;
}

public static bool ArealyRegistered(string nm)
{
return registrationDic.Keys.Contains(nm) || abbreviationDic.Keys.Contains(nm);
}

public static bool ArealyRegistered(Unit unt)
{
return registrationDic.Values.Where(i => i.Equals(unt)).Count() == 1;
}

internal static Unit ParseDirect(string unt)
{
//TODO: handle more complex units
var pat = @"(\S+)";
if (!System.Text.RegularExpressions.Regex.IsMatch(unt, pat))
throw new UnitConversionLibException(string.Format("Fomat mismatch:\r\n{0}", unt));

var mt = System.Text.RegularExpressions.Regex.Match(unt, pat);
var unit = mt.Groups[1].Value;
if (ArealyRegistered(unit))
return GetRegisteredUnit(unit);
```

```
throw new UnitConversionLibException(string.Format("Fomat mismatch:\r\n{0}", unt));
}

public static Unit Parse(string unt)
{
var prsr = new ReversePolishNotation();

prsr.Parse(unt);


var res= prsr.Evaluate();
return res;
//TODO: handle more complex units
var pat = @"(\S+)";
if (!System.Text.RegularExpressions.Regex.IsMatch(unt, pat))
throw new UnitConversionLibException(string.Format("Fomat mismatch:\r\n{0}", unt));

var mt = System.Text.RegularExpressions.Regex.Match(unt, pat);
var unit = mt.Groups[1].Value;
if (ArealyRegistered(unit))
return GetRegisteredUnit(unit);

throw new UnitConversionLibException(string.Format("Fomat mismatch:\r\n{0}", unt));
}

#region Operators

public static implicit operator Unit(string val)
{
var temp = Unit.Parse(val);
// code to convert from int to SampleClass...
return temp;
}

public static Unit operator *(Unit left, Unit right)
{
if (left.Scale != 1)
if (left.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

if (right.Scale != 1)
if (right.OverridedLocalName == null)
```

```
throw UnitConversionLibException.UnregedScalic;

var newDev = new UnitDev();

newDev.soorate.Add(new UnitBaseExpr(left.Copy()));
newDev.soorate.Add(new UnitBaseExpr(right.Copy()));

var buf = new Unit(newDev);

return buf;
}

public static bool operator ==(Unit left, Unit right)
{
return left.Equals(right);
}

public static bool operator !=(Unit left, Unit right)
{
return !left.Equals(right);
}

public static Unit operator /(Unit left, Unit right)
{
if (left.Scale != 1)
if (left.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

if (right.Scale != 1)
if (right.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

var newDev = new UnitDev();

newDev.soorate.Add(new UnitBaseExpr(left.Copy()));
newDev.makhraj.Add(new UnitBaseExpr(right.Copy()));

var buf = new Unit(newDev);
return buf;
}

public static Unit operator *(double sc, Unit unt)
```

```
{
if (unt.Scale != 1)
if (unt.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

if (sc <= 0)
throw UnitConversionLibException.ScaleNonPositive;


if (unt.Scale == 0)
throw UnitConversionLibException.DirectUnitCreated;

var newDev = new UnitDev();

newDev.soorate.Add(new UnitBaseExpr(unt.Copy()));

var buf = new Unit(newDev, sc);

return buf;
}

public static Unit operator *(Unit unt, double sc)
{
return sc*unt;
}

public static Unit operator /(double sc, Unit unt)
{
if (unt.Scale != 1)
if (unt.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

if (sc <= 0)
throw UnitConversionLibException.ScaleNonPositive;


if (unt.Scale == 0)
throw UnitConversionLibException.DirectUnitCreated;

var newDev = new UnitDev();

newDev.makhraj.Add(new UnitBaseExpr(unt.Copy()));
```

```
var buf = new Unit(newDev);

return sc*unt;
}

public static Unit operator /(Unit unt, double sc)
{
return (1/sc)*unt;
}

public static Unit operator ^(Unit unt, double pow)
{
if (unt.Scale != 1)
if (unt.OverridedLocalName == null)
throw UnitConversionLibException.UnregedScalic;

if (unt.Scale == 0)
throw UnitConversionLibException.DirectUnitCreated;

var newDev = new UnitDev();

newDev.soorate.Add(new UnitBaseExpr(unt.Copy(), pow));

var buf = new Unit(newDev);

return buf;
}

#endregion

#endregion

#endregion

#region Fields

internal static Dictionary<string, Unit> registrationDic = new Dictionary<string, Unit>();
internal static Dictionary<string, string> abbreviationDic = new Dictionary<string, string>();

public bool IsBase;
public string Name;
```

```
public string OverridedLocalName;
public double Scale;

public UnitDev dev;

public bool NotIsScalic;
/*{
get
{
if (this.dev == null)
throw UnitConversionLibException.DirectUnitCreated;

return !this.GetOrigins().IsScalic();
}
}*/

public string LocalCaption
{
get
{
//return this.LocalCaption;
return this.OverridedLocalName ?? GetLocalCaption().ToString();
}
}

public string GlobalCaption
{
get { return Unit.GetGlobalName(this); }
}

#endregion
}
}
```

UnitBaseExpr.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace UnitConversionLib
{
```

```
public class UnitBaseExpr
{
#region Constructors
/// <summary>
/// Initializes a new instance of the <see cref="T:System.Object"/> class.
/// </summary>
public UnitBaseExpr()
{
}

/// <summary>
/// Initializes a new instance of the <see cref="T:System.Object"/> class.
/// </summary>
public UnitBaseExpr(Unit @base, double pow = 1)
{
this._base = @base;
this.pow = pow;
}

#endregion
public Unit _base;
public double pow;

public UnitBaseExpr Copy()
{
return new UnitBaseExpr(_base, pow);
}
}
}
```

UnitConversionLibException.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;

namespace UnitConversionLib
{
public class UnitConversionLibException : Exception
{
```

```
#region Constructors

public UnitConversionLibException()
: base()
{
}

public UnitConversionLibException(string message)
: base(message)
{
}

public UnitConversionLibException(string message, Exception innerException)
: base(message, innerException)
{
}

public UnitConversionLibException(SerializationInfo info, StreamingContext context)
: base(info, context)
{
}

#endregion

#region Static Methods

public static void ThrowIf(bool condition, string message = "", Exception inner = null)
{
if (condition)
throw new UnitConversionLibException(message, inner);
}

public static void Throw(string message = "", Exception inner = null)
{
throw new UnitConversionLibException(message, inner);
}

public static UnitConversionLibException UnregedScalic
{
get
{
return new UnitConversionLibException("Unregistered Scalic Unit");
```

```
}
}

public static UnitConversionLibException ScaleNonPositive
{
get
{
return new UnitConversionLibException("No Negative or zero Scale is allowed");
}
}

public static UnitConversionLibException DirectUnitCreated
{
get
{
return new UnitConversionLibException("No Direct Created Unit Allowed");
}
}

public static UnitConversionLibException InconsistentUnits
{
get
{
return new UnitConversionLibException("Inconcsistent units detected while converting");
}
}

#endregion

}

public class UnitParsingException : UnitConversionLibException
{
public UnitParsingException()
{
}

public UnitParsingException(string message)
: base(message)
{
}
```

```
public UnitParsingException(string message, Exception innerException)
: base(message, innerException)
{
}

public UnitParsingException(SerializationInfo info, StreamingContext context)
: base(info, context)
{
}

public int StartCharacter;

public string UnknownPhraze;
}
}
```

UnitDev.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace UnitConversionLib
{
public class UnitDev
{
public List<UnitBaseExpr> soorate = new List<UnitBaseExpr>();
public List<UnitBaseExpr> makhraj = new List<UnitBaseExpr>();

#region Methods
public UnitDev Copy()
{
var buf = new UnitDev();

buf.soorate.AddRange(this.soorate.Select(i => i.Copy()));
buf.makhraj.AddRange(this.makhraj.Select(i => i.Copy()));

return buf;
}

public BaseUnitDev GetOrigins()
{
```

```
var so = this.soorate.Select(i =>
{
var t = i._base.GetOrigins();
t.ApplyPower(i.pow);
return t;
});

var mo = this.makhraj.Select(i =>
{
var t = i._base.GetOrigins();
t.ApplyPower(i.pow);
return t;
});

var st = so.SelectMany(i => i.soorate).ToList();
st.AddRange(mo.SelectMany(i => i.makhraj));

var mt = so.SelectMany(i => i.makhraj).ToList();
mt.AddRange(mo.SelectMany(i => i.soorate));

var buf = new BaseUnitDev();
buf.soorate.AddRange(st);
buf.makhraj.AddRange(mt);

buf.Simplice();

return buf;
}
#endregion
}
}
```

.NETFramework,Version=v4.6.1.AssemblyAttributes.cs

```
// <autogenerated />
using System;
using System.Reflection;
[assembly:
global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETFramework,Version=v4.6.1",
FrameworkDisplayName = ".NET Framework 4.6.1")]
```

AssemblyInfo.cs

```
using System.Reflection;
```

```
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("UnitConversionLib")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Microsoft")]
[assembly: AssemblyProduct("UnitConversionLib")]
[assembly: AssemblyCopyright("Copyright © Microsoft 2011")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("14608bd1-a04c-4aa7-b69a-bb00734a5c0d")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyFileVersion("0.14.0.01")]
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
```

```
namespace UnitConversionLib.Example
{
class Program
{
static void Main(string[] args)
{
Ex2();
Console.WriteLine("==== ex1, working with units");
Console.WriteLine("Checking the Newton (N) unit of force.");

Unit unt0 = Unit.GetRegisteredUnit("N");                    //From registered library

var str2 = "N";
var str3 = "kg*m/sec^2";
var str4 = "kg*m/s^(ton/kg/500)";
var str5 = "kg*m/s^(kg/kg*2)";



Unit unt1 = Unit.Parse("N");        //Parsing from string - simple format
Unit unt2 = str2;                   //Explicit conversion
Unit unt3 = Unit.Parse(str3);       //Parsing from string - from expression (1)
Unit unt4 = Unit.Parse(str4);       //Parsing from string - from expression (2)
Unit unt5 = Unit.Parse(str5);       //Parsing from string - from expression (3)
Unit unt6 = Unit.Parse("kg*m") / (Unit.Parse("sec") ^ 2);   //Parsing from string - from expression
(4)

Console.WriteLine();
Console.WriteLine("- {0} {1} equal to {2}", unt1, unt1 == unt2 ? "is" : "is not", unt2);
Console.WriteLine();

Console.WriteLine("- {0} {1} even equal to these:\n\t{3},\n\t{4},\n\t{5}",unt1, (unt1 == unt2 && unt1
== unt3 && unt1 == unt4 & unt1 == unt5) ? "is" : "is not", str2, str3, str4, str5);

var kmph = Unit.Parse("km/hour");
Unit.Register(ref kmph, "kmph");
var D = Measurable.Parse("100 km");
var T = Measurable.Parse("60 min");

Console.WriteLine("");
Console.WriteLine("==== ex2, the driving car");
Console.WriteLine("Consider a car who was drived about 100 km in 60 min (D = '{0}',T = '{1}')", D,
```

```
T);

Console.WriteLine("it can be said the car also is derived '{0:0.00}' in '{1:0.0}'", D.ConvertTo("mi"),
T.ConvertTo("hour"));
Console.WriteLine("or even '{0:0.00}' in '{1:0.0}'", D.ConvertTo("foot"), T.ConvertTo("sec"));
var avgSpeed = D/T;
Console.WriteLine("Finally the average speed is D/T='{0:0.0}'='{1:0.0}'='{2:0.0}'", avgSpeed,
avgSpeed.ConvertTo("km / hour"), avgSpeed.ConvertTo("kmph"));

Console.WriteLine("");
Console.WriteLine("** Check the code of this console application for more samples (check the
code inside CodeExample() method)...");

Console.ReadKey();
}

static void CodeExample()
{
//Creating and parsing units
Unit unt0 = Unit.GetRegisteredUnit("N");                    //From registered library
Unit unt1 = Unit.Parse("N");                                //Parsing from string - simple format
Unit unt2 = "N";                                            //Explicit conversion
Unit unt3 = Unit.Parse("kg*m/sec^2");                       //Parsing from string - from expression (1)
Unit unt4 = Unit.Parse("kg*m/s^(ton/kg/500)");              //Parsing from string - from expression (2)
Unit unt5 = Unit.Parse("kg*m/s^(kg/kg*2)");                 //Parsing from string - from expression (3)
Unit unt6 = Unit.Parse("kg*m") / (Unit.Parse("sec") ^ 2);   //Parsing from string - from expression
(4)

Unit kg = Unit.Parse("kg");
Unit m = Unit.Parse("m");
Unit sec = Unit.Parse("sec");

Unit unt7 = kg * m / (sec ^ 2);                             //Evaluating thro runtime

Debug.Assert(unt0 == unt1 && unt1 == unt2 && unt2 == unt3 && unt3 == unt4 && unt4 == unt5
&& unt5 == unt6 && unt6 == unt7);

Unit unt8 = Unit.Parse("kg*m/s^(kg)");                      //will raise error, a unit cannot be placed in
power

//In Action I:
Measurable meu0 = Measurable.Parse("100 N");
```

```
Measurable meu1 = "100 N";
Measurable meu2 = new Measurable(unt0, 100);

Debug.Assert(meu0 == meu1);

Measurable meu3 = meu0.ConvertTo("kgf");
Debug.Assert(meu3.Amount == 1000);
Debug.Assert(meu3.Unit == "kgf");

//In action II, calculating average acceleration

Measurable deltax = "100 m";
Measurable deltaT = "10 sec";
Measurable scUnit = "2 kg/kg";

Measurable avgAccl = deltax / (deltaT ^ scUnit);          //which is 1 m/sec^2

Debug.Assert(avgAccl.Amount == 1);
Debug.Assert(avgAccl.Unit == "m/  sec^2");


//In action III: creating new units
Unit bit = Unit.CreateNew("b");
Unit B = bit * 8;
Unit.Register(ref B, "B");
Unit KB = B * 1024;
Unit.Register(ref KB, "KB");
Unit MB = KB * 1024;
Unit.Register(ref MB, "MB");
Unit GB = MB * 1024;
Unit.Register(ref GB, "GB");
Unit TB = GB * 1024;
Unit.Register(ref TB, "TB");

Unit bps = bit / Unit.GetRegisteredUnit("sec");
Unit.Register(ref bps, "bps");
Measurable smb = "100 MB/sec";
Measurable sb = smb.ConvertTo("bps");// 100 MB = 100*1024*1024*8 b = 838860800 b

Debug.Assert(sb.Amount == 838860800);
Debug.Assert(sb.Unit == bit / "sec");
```

```
}

static void Ex2()
{
var D = Measurable.Parse("100 m");
var T = Measurable.Parse("10 sec");

var V = D/T;

Console.WriteLine("{0}",V); // will show 10 m/sec

Console.WriteLine("{0}", V.ConvertTo("foot/hour")); // will show 118109.999 foot/hr
}
}
}
```

AssemblyInfo.cs

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("UnitConversionLib.Example")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("UnitConversionLib.Example")]
[assembly: AssemblyCopyright("Copyright ©  2014")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("5671917b-1456-4c5a-8ff7-e34142969776")]

// Version information for an assembly consists of the following four values:
```

```
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyFileVersion("0.14.0.01")]
```

AiqFileReader.cs

```
using System;
using System.Text;
using System.Runtime.InteropServices;


namespace AiqFileReaderDotNet
{
/// <summary>
/// Summary description for AiqFileReaderDotNet.
/// </summary>
public class AiqFileReader : IDisposable
{
#region Members

private bool m_isDisposed = false;

private static IntPtr dllHandle = IntPtr.Zero;

private Int32 m_session = 0;
private Int32 m_status = 0;

#endregion

#region Properties

private const int STRING_BUILER_SIZE = 256; // AiqFileReader DLL allocates 256 bytes

private StringBuilder m_description = new StringBuilder(STRING_BUILER_SIZE);
public string Description
```

```
{
get { return m_description.ToString(); }
}

private StringBuilder m_fileVersion = new StringBuilder(STRING_BUILER_SIZE);
public string FileVersion
{
get { return m_fileVersion.ToString(); }
}

private StringBuilder m_packagerPartNumber = new StringBuilder(STRING_BUILER_SIZE);
public string PackagerPartNumber
{
get { return m_packagerPartNumber.ToString(); }
}

private float m_packagerVersion;
public float PackagerVersion
{
get { return m_packagerVersion; }
}

private Int32 m_numberOfSamples;
public Int32 NumberOfSamples
{
get { return m_numberOfSamples; }
}

private double m_samplingRate;
public double SamplingRate
{
get
{
return m_samplingRate;
}
}

private double m_signalBandwidth;
public double SignalBandwidth
{
get { return m_signalBandwidth; }
}
```

```
private Int32 m_modulationId;
public Int32 ModulationID
{
get { return m_modulationId; }
}

private Int32 m_rms;
public Int32 Rms
{
get { return m_rms; }
}

private float m_relativeRms;
public float RelRms
{
get { return m_relativeRms; }
}

private float m_crestFactor;
public float CrestFactor
{
get { return m_crestFactor; }
}

private StringBuilder m_levelMode = new StringBuilder(STRING_BUILER_SIZE);
public string LevelMode
{
get { return m_levelMode.ToString(); }
}

private Int32 m_alcBandwidth;
public Int32 AlcBandwidth
{
get { return m_alcBandwidth; }
}

private UInt16 m_markerDelay;
public UInt16 MarkerDelay
{
get { return m_markerDelay; }
}
```

```
private Int32 m_upPowerRampType;
public Int32 UpPowerRampType
{
get { return m_upPowerRampType; }
}

private Int32 m_downPowerRampType;
public Int32 DownPowerRampType
{
get { return m_downPowerRampType; }
}

private float m_upPowerRampTime;
public float UpPowerRampTime
{
get { return m_upPowerRampTime; }
}

private float m_downPowerRampTime;
public float DownPowerRampTime
{
get { return m_downPowerRampTime; }
}

private Int32 m_marker1Assignment;
public Int32 Marker1Assignment
{
get { return m_marker1Assignment; }
}

private Int32 m_marker2Assignment;
public Int32 Marker2Assignment
{
get { return m_marker2Assignment; }
}

private Int32 m_marker3Assignment;
public Int32 Marker3Assignment
{
get { return m_marker3Assignment; }
}
```

```
private Int32 m_marker4Assignment;
public Int32 Marker4Assignment
{
get { return m_marker4Assignment; }
}

private float[] m_i;
public float[] I
{
get { return m_i; }
}

private float[] m_q;
public float[] Q
{
get { return m_q; }
}

#endregion

#region Constructor/Destructor

public AiqFileReader(string aiqFilePath)
{
LoadLibrary(AIQ_READER_DLL);

ReadAiqFile(aiqFilePath);
}

public AiqFileReader(string aiqReaderDllPath, string aiqFilePath)
{
LoadLibrary(aiqReaderDllPath);

ReadAiqFile(aiqFilePath);
}

~AiqFileReader()
{
Dispose(false);
}
```

```
#endregion

#region Public Methods

public Int32 ReadIQ(Int32 offset, Int32 length, float[] iData, float[] qData)
{
Int32 numberOfSamplesRead = 0;

if(iData.Length < length || qData.Length < length)
throw new Exception("Number of samples requested exceeds output array size");

if((offset + length) > m_numberOfSamples)
throw new Exception("Number of IQ samples requested exceeds number available");

m_status = AiqFileReader_ReadIQ(m_session, out numberOfSamplesRead, iData, qData, length,
offset);
CheckError();

return numberOfSamplesRead;
}

public Int32 ReadIQWithMarkerData(Int32 offset, Int32 length, Int32[] iqData)
{
Int32 numberOfSamplesRead = 0;

if(iqData.Length < length)
throw new Exception("Number of samples requested exceeds output array size");

if((offset + length) > m_numberOfSamples)
throw new Exception("Number of IQ samples requested exceeds number available");

m_status = AiqFileReader_ReadIQ_WithMarker(m_session, out numberOfSamplesRead, iqData,
length, offset);
CheckError();

return numberOfSamplesRead;
}

#endregion

#region Protected Methods
```

```
protected virtual void Dispose(bool disposing)
{
if(!m_isDisposed)
{
if(disposing)
{
// Dispose of managed objects here
//
// Method not being called from destructor, so OK to reference other objects here

// Nothing to dispose of!
}

// Release unmanaged resources
if(m_session != 0)
{
m_status = AiqFileReader_Close(m_session);
m_session = 0;
}
}
m_isDisposed = true;
}

#endregion

#region Private Methods

[DllImport("Kernel32.dll")]
private extern static IntPtr LoadLibrary(string path);

private static void LoadDLL(String path)
{
if (dllHandle != IntPtr.Zero)
return;

if (path == null)
{
#if DEBUG
path = @"..\..\..\Dll\Debug\AiqFileReader.dll";
#else
path = @"..\..\..\Dll\Release\AiqFileReader.dll";
#endif
```

```
dllHandle = LoadLibrary(path);
}
else
{
dllHandle = LoadLibrary(path);
}

if (dllHandle == IntPtr.Zero)
throw new Exception(String.Format("Unable to load {0}", path));
}

private void ReadAiqFile(string aiqFilePath)
{
// Open AIQ file
this.m_status = AiqFileReader_Open(out this.m_session, aiqFilePath);
CheckError("Unable to open AIQ file");

// Read header data
this.m_status = AiqFileReader_ReadHeaderMkrData(this.m_session);
CheckError();

// Get Description
this.m_status = AiqFileReader_GetDescription(this.m_session, this.m_description,
this.m_description.Capacity);
CheckError();

// Get File Version
this.m_status = AiqFileReader_FileVersion_Get(this.m_session, this.m_fileVersion);
CheckError();

// Get Packager Part No
this.m_status = AiqFileReader_Packager_PartNumber_Get(this.m_session,
this.m_packagerPartNumber);
CheckError();

// Get Packager Version
this.m_status = AiqFileReader_Packager_Version_Get(this.m_session, out
this.m_packagerVersion);
CheckError();

// Get Modulation ID
this.m_status = AiqFileReader_ModulationID_Get(this.m_session, out this.m_modulationId);
```

```
CheckError();

// Get Signal Bandwidth
this.m_status = AiqFileReader_SignalBandwidth_Get(this.m_session, out
this.m_signalBandwidth);
CheckError();

// Get RMS
this.m_status = AiqFileReader_Rms_Get(this.m_session, out this.m_rms);
CheckError();

// Get Relative RMS
this.m_status = AiqFileReader_RelRms_Get(this.m_session, out this.m_relativeRms);
CheckError();

// Get Crest Factor
this.m_status = AiqFileReader_CrestFactor_Get(this.m_session, out this.m_crestFactor);
CheckError();

// Get Level Mode
this.m_status = AiqFileReader_LvlMode_Get(this.m_session, this.m_levelMode);
CheckError();

// Get ALC Bandwidth
this.m_status = AiqFileReader_AlcBandwidth_Get(this.m_session, out this.m_alcBandwidth);
CheckError();

// Get Marker Delay
this.m_status = AiqFileReader_Mkr_delay_Get(this.m_session, out this.m_markerDelay);
CheckError();

// Get Up Power Ramp Type
this.m_status = AiqFileReader_UpPwrRampType_Get(this.m_session, out
this.m_upPowerRampType);
CheckError();

// Get Down Power Ramp Type
this.m_status = AiqFileReader_DownPwrRampType_Get(this.m_session, out
this.m_downPowerRampType);
CheckError();

// Get Up Power Ramp Time
```

```
this.m_status = AiqFileReader_UpPwrRampTime_Get(this.m_session, out
this.m_upPowerRampTime);
CheckError();

// Get Down Power Ramp Time
this.m_status = AiqFileReader_DownPwrRampTime_Get(this.m_session, out
this.m_downPowerRampTime);
CheckError();

// Get Marker 1 Assignment
this.m_status = AiqFileReader_Mkr1_Assignement_Get(this.m_session, out
this.m_marker1Assignment);
CheckError();

// Get Marker 2 Assignment
this.m_status = AiqFileReader_Mkr2_Assignement_Get(this.m_session, out
this.m_marker2Assignment);
CheckError();

// Get Marker 3 Assignment
this.m_status = AiqFileReader_Mkr3_Assignement_Get(this.m_session, out
this.m_marker3Assignment);
CheckError();

// Get Marker 4 Assignment
this.m_status = AiqFileReader_Mkr4_Assignement_Get(this.m_session, out
this.m_marker4Assignment);
CheckError();

// Get number of samples & allocate storage for IQ data
this.m_status = AiqFileReader_NumSamples_Get(this.m_session, out this.m_numberOfSamples);
CheckError();
this.m_i = new float[m_numberOfSamples];
this.m_q = new float[m_numberOfSamples];

// Get Sampling rate
this.m_status = AiqFileReader_SamplingRate_Get(this.m_session, out this.m_samplingRate);
CheckError();

// Read IQ data
this.ReadIQ(0, this.m_numberOfSamples, this.m_i, this.m_q);
}
```

```
private void CheckError()
{
CheckError("Unable to process AIQ file");
}

private void CheckError(string baseErrorMsg)
{
if(m_status != 0)
{
string errMsg = baseErrorMsg + " - " + m_status.ToString("X");

throw new Exception(errMsg);
}
}

#endregion

#region IDisposable Members

public void Dispose()
{
Dispose(true);

// Prevent garbage collector from finalizing this object (already done)
GC.SuppressFinalize(this);
}

#endregion

#region AiqFileReader DLL Imports

private const string AIQ_READER_DLL = @"AiqFileReader.dll";

//
// These DLL imports have been declared public to enable direct access to the DLL function calls.
//

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_Open(out Int32 ID, string AiqFileName);

[DllImport(AIQ_READER_DLL)]
```

```
public static extern Int32 AiqFileReader_Close(Int32 ID);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_ReadIQ(Int32 ID, out Int32 NumSamplesRead, float[]
IData, float[] QData, Int32 NumSamples, Int32 SampleOffset);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_ReadIQ_WithMarker(Int32 ID, out Int32
NumSamplesRead, Int32[] IQData, Int32 NumSamples, Int32 SampleOffset);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_ReadHeaderMkrData(Int32 ID);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_GetDescription(Int32 ID, StringBuilder Description, Int32
NumChars);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_SamplingRate_Get(Int32 ID, out double SamplingRate);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_NumSamples_Get(Int32 ID, out Int32 NumSamples);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_SignalBandwidth_Get(Int32 ID, out double
SignalBandwidth);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_ModulationID_Get(Int32 ID, out Int32 ModID);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_FileVersion_Get(Int32 ID, StringBuilder FileVersion);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_Packager_PartNumber_Get(Int32 ID, StringBuilder
PackagerPartNumber);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_Packager_Version_Get(Int32 ID, out float
PackagerVersion);

[DllImport(AIQ_READER_DLL)]
```

```
public static extern Int32 AiqFileReader_Rms_Get(Int32 ID, out Int32 rms);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_RelRms_Get(Int32 ID, out float relRms);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_CrestFactor_Get(Int32 ID, out float crestFactor);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_LvlMode_Get(Int32 ID, StringBuilder lvlMode);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_AlcBandwidth_Get(Int32 ID, out Int32 alcBandwidth);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_Mkr_delay_Get(Int32 ID, out UInt16 mkrDelay);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_UpPwrRampType_Get(Int32 ID, out Int32
upPwrRampType);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_DownPwrRampType_Get(Int32 ID, out Int32
dwnPwrRampType);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_UpPwrRampTime_Get(Int32 ID, out float
upPwrRampTime);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_DownPwrRampTime_Get(Int32 ID, out float
dwnPwrRampTime);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_Mkr1_Assignement_Get(Int32 ID, out Int32
Mkr1Assignment);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_Mkr2_Assignement_Get(Int32 ID, out Int32
Mkr2Assignment);

[DllImport(AIQ_READER_DLL)]
```

```
public static extern Int32 AiqFileReader_Mkr3_Assignement_Get(Int32 ID, out Int32
Mkr3Assignment);

[DllImport(AIQ_READER_DLL)]
public static extern Int32 AiqFileReader_Mkr4_Assignement_Get(Int32 ID, out Int32
Mkr4Assignment);

#endregion
}
}
```

Extension.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net.Mail;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;

namespace WaveformConverterControls
{
public static class Extensions
{
//Extension method for MailMessage to save to a file on disk
public static void Save(this MailMessage message, string filename, bool addUnsentHeader = true)
{
using (var filestream = File.Open(filename, FileMode.Create))
{
if (addUnsentHeader)
{
var binaryWriter = new BinaryWriter(filestream);
//Write the Unsent header to the file so the mail client knows this mail must be presented in "New
message" mode
binaryWriter.Write(System.Text.Encoding.UTF8.GetBytes("X-Unsent: 1" +
Environment.NewLine));
}

var assembly = typeof(SmtpClient).Assembly;
var mailWriterType = assembly.GetType("System.Net.Mail.MailWriter");
```

```
// Get reflection info for MailWriter contructor
var mailWriterContructor = mailWriterType.GetConstructor(BindingFlags.Instance |
BindingFlags.NonPublic, null, new[] { typeof(Stream) }, null);

// Construct MailWriter object with our FileStream
var mailWriter = mailWriterContructor.Invoke(new object[] { filestream });

// Get reflection info for Send() method on MailMessage
var sendMethod = typeof(MailMessage).GetMethod("Send", BindingFlags.Instance |
BindingFlags.NonPublic);

sendMethod.Invoke(message, BindingFlags.Instance | BindingFlags.NonPublic, null, new object[] {
mailWriter, true, true }, null);

// Finally get reflection info for Close() method on our MailWriter
var closeMethod = mailWriter.GetType().GetMethod("Close", BindingFlags.Instance |
BindingFlags.NonPublic);

// Call close method
closeMethod.Invoke(mailWriter, BindingFlags.Instance | BindingFlags.NonPublic, null, new object[]
{ }, null);
}
}
}
}
```

INotifyBase.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WaveformConverterControls
{
public abstract class INotifyBase : INotifyPropertyChanged
{
#region INotifyPropertyChanged
```

```
public event PropertyChangedEventHandler PropertyChanged;

/// <summary>
/// Create the OnPropertyChanged method to raise the event.
/// </summary>
/// <param name="name">Name of the property that's just been updated.</param>
protected void OnPropertyChanged(string name)
{
PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null)
{
handler(this, new PropertyChangedEventArgs(name));
}
}

#endregion INotifyPropertyChanged
}
}
```

ListBoxBehavior.cs

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;

namespace WaveformConverterControls
{
public class ListBoxBehavior
{
static readonly Dictionary<ListBox, Capture> Associations =
new Dictionary<ListBox, Capture>();

public static bool GetScrollOnNewItem(DependencyObject obj)
{
return (bool)obj.GetValue(ScrollOnNewItemProperty);
}
```

```
public static void SetScrollOnNewItem(DependencyObject obj, bool value)
{
obj.SetValue(ScrollOnNewItemProperty, value);
}

public static readonly DependencyProperty ScrollOnNewItemProperty =
DependencyProperty.RegisterAttached(
"ScrollOnNewItem",
typeof(bool),
typeof(ListBoxBehavior),
new UIPropertyMetadata(false, OnScrollOnNewItemChanged));

public static void OnScrollOnNewItemChanged(
DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
var listBox = d as ListBox;
if (listBox == null) return;
bool oldValue = (bool)e.OldValue, newValue = (bool)e.NewValue;
if (newValue == oldValue) return;
if (newValue)
{
listBox.Loaded += ListBox_Loaded;
listBox.Unloaded += ListBox_Unloaded;
var itemsSourcePropertyDescriptor = TypeDescriptor.GetProperties(listBox)["ItemsSource"];
itemsSourcePropertyDescriptor.AddValueChanged(listBox, ListBox_ItemsSourceChanged);
}
else
{
listBox.Loaded -= ListBox_Loaded;
listBox.Unloaded -= ListBox_Unloaded;
if (Associations.ContainsKey(listBox))
Associations[listBox].Dispose();
var itemsSourcePropertyDescriptor = TypeDescriptor.GetProperties(listBox)["ItemsSource"];
itemsSourcePropertyDescriptor.RemoveValueChanged(listBox, ListBox_ItemsSourceChanged);
}
}

private static void ListBox_ItemsSourceChanged(object sender, EventArgs e)
{
var listBox = (ListBox)sender;
```

```
if (Associations.ContainsKey(listBox))
Associations[listBox].Dispose();
Associations[listBox] = new Capture(listBox);
}

static void ListBox_Unloaded(object sender, RoutedEventArgs e)
{
var listBox = (ListBox)sender;
if (Associations.ContainsKey(listBox))
Associations[listBox].Dispose();
listBox.Unloaded -= ListBox_Unloaded;
}

static void ListBox_Loaded(object sender, RoutedEventArgs e)
{
var listBox = (ListBox)sender;
var incc = listBox.Items as INotifyCollectionChanged;
if (incc == null) return;
listBox.Loaded -= ListBox_Loaded;
Associations[listBox] = new Capture(listBox);
}

class Capture : IDisposable
{
private readonly ListBox listBox;
private readonly INotifyCollectionChanged incc;

public Capture(ListBox listBox)
{
this.listBox = listBox;
incc = listBox.ItemsSource as INotifyCollectionChanged;
if (incc != null)
{
incc.CollectionChanged += incc_CollectionChanged;
}
}

void incc_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
{
if (e.Action == NotifyCollectionChangedAction.Add)
{
listBox.ScrollIntoView(e.NewItems[0]);
```

```
listBox.SelectedItem = e.NewItems[0];
}
}

public void Dispose()
{
if (incc != null)
incc.CollectionChanged -= incc_CollectionChanged;
}
}
}
}
```

MainWindowViewModel.cs

```
using AiqFileReaderDotNet;
//using MT.Template.GUI;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Diagnostics;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Net.Mail;
using System.Runtime.InteropServices;
using System.Security.Cryptography;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Forms.DataVisualization.Charting;
using System.Windows.Input;
using System.Windows.Threading;

namespace WaveformConverterControls
{
/// <summary>
/// An interface implementing the Supervising Controller pattern that allows the View to have some controls created dynamically.
```

```
/// </summary>
public interface IView
{
/// <summary>
/// Sets up the chart
/// </summary>
/// <param name="Title"></param>
/// <param name="XaxisTitle"></param>
/// <param name="YaxisTitle"></param>
void SetupChart(string Title, string XaxisTitle, string Xaxis2Title, string YaxisTitle, double
XaxisMinimum, double Xaxis2Minimum, double XaxisMaximum, double Xaxis2Maximum);

void DrawSeries(double[] xValues, double[] results, int seriesIndex, AxisType axisType);
}

public class MainWindowViewModel : INotifyPropertyChanged
{
//[DllImport("C:\\Analysis\\MT Generation\\Convert AIQ to MTWFM\\Convert AIQ to
MTWFM\\bin\\Debug\\PackageToAiq.dll", ExactSpelling = true, CallingConvention =
CallingConvention.Cdecl)]
[DllImport("PackageToAiq.dll", ExactSpelling = true, CallingConvention =
CallingConvention.Cdecl)]
public static extern int PackageAiqWaveform(string FileNameIq, string FileNameMarker, string
FileNameOutput, string Description, float SampleRateHz);

#region Private Variables

private Format? radioFormat;
private float? crestFactor;
private float? dutyCycle;
private int? alcBandwidth;
private int? numberOfSamples;
private double? totalTime;
private float? relRms;
private int? rms;
private double? samplingRate;
private double? signalBandwidth;
private string description;
private double? offset;
private double? measLength;
private double? stepLength;
private int? numSteps;
```

```
private double? subcarrierSpacing;
private CyclicPrefix? cycPrefix;
private bool? transformPrecoding;
private PUSCHModulation? puschMod;
private string puschSlots;
private string puschPRB;
private double? guardInterval;
private int? mcs;
private DataRate? dRate;
private LTFCompType? ltfType;
private int? dataSymbols;
private RadioConfig? rConfig;
private PhysicalLayerSubtype? physLayerSubtype;
private LTEDownlinkMode? dlMode;
private int? uldlConfigIndex;
private int? pdcchNumSymbol;
private List<double> Idata = new List<double>();
private List<double> Qdata = new List<double>();
private List<double> Power = new List<double>();
private List<int> Marker1 = new List<int>();
private List<int> Marker2 = new List<int>();
private List<int> Marker3 = new List<int>();
private List<int> Marker4 = new List<int>();
private string InputFileName { get; set; }
private AiqFileReader currentLoadedAiqFile;
private MTWaveform m_currentMTWaveform;


#endregion Private Variables

#region Constructor

/// <summary>
/// Constructor for the MainWindowViewModel
/// </summary>
public MainWindowViewModel()
{
ComboboxRadioFormatList = new ObservableCollection<KeyValuePair<Format?, string>>
{
new KeyValuePair<Format?, string>(null, ""),
new KeyValuePair<Format?, string>(Format.Undefined, "Undefined"),
new KeyValuePair<Format?, string>(Format.C2K, "C2K"),
```

```
new KeyValuePair<Format?, string>(Format.GSM_EDGE, "GSM"),
new KeyValuePair<Format?, string>(Format.LTE_Downlink, "LTE Downlink"),
new KeyValuePair<Format?, string>(Format.LTE_Uplink_FDD, "LTE Uplink FDD"),
new KeyValuePair<Format?, string>(Format.LTE_Uplink_TDD, "LTE Uplink TDD"),
new KeyValuePair<Format?, string>(Format.NR, "NR"),
new KeyValuePair<Format?, string>(Format.TDSCDMA, "TDSCDMA"),
new KeyValuePair<Format?, string>(Format.WCDMA, "WCDMA"),
new KeyValuePair<Format?, string>(Format.WLAN_AC, "WLAN AC"),
new KeyValuePair<Format?, string>(Format.WLAN_AG, "WLAN AG"),
new KeyValuePair<Format?, string>(Format.WLAN_AX, "WLAN AX"),
new KeyValuePair<Format?, string>(Format.WLAN_B, "WLAN B"),
new KeyValuePair<Format?, string>(Format.WLAN_BE, "WLAN BE"),
new KeyValuePair<Format?, string>(Format.WLAN_J, "WLAN J"),
new KeyValuePair<Format?, string>(Format.WLAN_N, "WLAN N"),
new KeyValuePair<Format?, string>(Format.WLAN_P, "WLAN P"),
new KeyValuePair<Format?, string>(Format._1xEvDo, "1xEvDo")
};
ComboboxRadioConfigList = new ObservableCollection<KeyValuePair<RadioConfig?, string>>
{
new KeyValuePair<RadioConfig?,string>(null,""),
new KeyValuePair<RadioConfig?, string>(RadioConfig.Undefined, "Undefined"),
new KeyValuePair<RadioConfig?, string>(RadioConfig.RC1, "RC1"),
new KeyValuePair<RadioConfig?, string>(RadioConfig.RC2, "RC2"),
new KeyValuePair<RadioConfig?, string>(RadioConfig.RC3, "RC3"),
new KeyValuePair<RadioConfig?, string>(RadioConfig.RC4, "RC4"),
};
ComboboxLTEDownlinkModeList = new
ObservableCollection<KeyValuePair<LTEDownlinkMode?, string>>
{
new KeyValuePair<LTEDownlinkMode?,string>(null,""),
new KeyValuePair<LTEDownlinkMode?, string>(LTEDownlinkMode.Undefined, "Undefined"),
new KeyValuePair<LTEDownlinkMode?, string>(LTEDownlinkMode.FDD, "FDD"),
new KeyValuePair<LTEDownlinkMode?, string>(LTEDownlinkMode.TDD, "TDD"),
};
ComboboxSubcarrierSpacingList = new ObservableCollection<double?>
{
null,double.NaN,15,30,60,
};
ComboboxCyclicPrefixList = new ObservableCollection<KeyValuePair<CyclicPrefix?, string>>
{
new KeyValuePair<CyclicPrefix?,string>(null,""),
new KeyValuePair<CyclicPrefix?, string>(CyclicPrefix.Undefined, "Undefined"),
```

```
new KeyValuePair<CyclicPrefix?, string>(CyclicPrefix.Normal, "Normal"),
new KeyValuePair<CyclicPrefix?, string>(CyclicPrefix.Extended, "Extended"),
};
ComboboxTransformPrecodingList = new ObservableCollection<bool?>
{
null,true,false,
};
ComboboxPUSCHModList = new ObservableCollection<KeyValuePair<PUSCHModulation?,
string>>
{
new KeyValuePair<PUSCHModulation?,string>(null,""),
new KeyValuePair<PUSCHModulation?, string>(PUSCHModulation.Undefined, "Undefined"),
new KeyValuePair<PUSCHModulation?, string>(PUSCHModulation.Pi_2_BPSK, "Pi/2 BPSK"),
new KeyValuePair<PUSCHModulation?, string>(PUSCHModulation.QPSK, "QPSK"),
new KeyValuePair<PUSCHModulation?, string>(PUSCHModulation._16QAM, "16 QAM"),
new KeyValuePair<PUSCHModulation?, string>(PUSCHModulation._64QAM, "64 QAM"),
new KeyValuePair<PUSCHModulation?, string>(PUSCHModulation._256QAM, "256 QAM"),
};
ComboboxGuardIntervalList = new ObservableCollection<double?>
{
null,double.NaN,0.8,1.6,3.2,
};
ComboboxMCSList = new ObservableCollection<int?>
{
null,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,
};
ComboboxDataRateList = new ObservableCollection<KeyValuePair<DataRate?, string>>
{
new KeyValuePair<DataRate?,string>(null,""),
new KeyValuePair<DataRate?, string>(DataRate.Undefined, "Undefined"),
new KeyValuePair<DataRate?, string>(DataRate._1Mbps, "1 Mbps"),
new KeyValuePair<DataRate?, string>(DataRate._2Mbps, "2 Mbps"),
new KeyValuePair<DataRate?, string>(DataRate._5_5Mbps, "5.5 Mbps"),
new KeyValuePair<DataRate?, string>(DataRate._11Mbps, "11 Mbps"),
};
ComboboxLTFCompTypeList = new ObservableCollection<KeyValuePair<LTFCompType?,
string>>
{
new KeyValuePair<LTFCompType?,string>(null,""),
new KeyValuePair<LTFCompType?, string>(LTFCompType.Undefined, "Undefined"),
new KeyValuePair<LTFCompType?, string>(LTFCompType._4x, "4x"),
new KeyValuePair<LTFCompType?, string>(LTFCompType._2x, "2x"),
```

```
new KeyValuePair<LTFCompType?, string>(LTFCompType._1x, "1x"),
};
ComboboxPhysLayerSubtypeList = new
ObservableCollection<KeyValuePair<PhysicalLayerSubtype?, string>>
{
new KeyValuePair<PhysicalLayerSubtype?,string>(null,""),
new KeyValuePair<PhysicalLayerSubtype?, string>(PhysicalLayerSubtype.Undefined,
"Undefined"),
new KeyValuePair<PhysicalLayerSubtype?, string>(PhysicalLayerSubtype.Subtype0, "0"),
new KeyValuePair<PhysicalLayerSubtype?, string>(PhysicalLayerSubtype.Subtype1, "1"),
new KeyValuePair<PhysicalLayerSubtype?, string>(PhysicalLayerSubtype.Subtype2, "2"),
};

MetadataEditingEnabled = false;
}
~MainWindowViewModel()
{
//Console.WriteLine("Waveform Control Finalized...");
}

#endregion Constructor

#region Public Properties

/// <summary>
/// An instance of the IView interface. This controls access to the VIEW from the VIEWMODEL
using the supervising controller pattern.
/// </summary>
private IView _view;
public IView View
{
get { return _view; }
set { _view = value; OnPropertyChanged("View"); }
}

// Visibility options for each radio format
private Visibility nrVis = Visibility.Collapsed;
private Visibility wlanVis = Visibility.Collapsed;
private Visibility c2kVis = Visibility.Collapsed;
private Visibility evdoVis = Visibility.Collapsed;
private Visibility lteDownlinkVis = Visibility.Collapsed;
public Visibility NRVis
```

```
{
get
{
return nrVis;
}
set
{
nrVis = value;
OnPropertyChanged("NRVis");
}
}
public Visibility WLANVis
{
get
{
return wlanVis;
}
set
{
wlanVis = value;
OnPropertyChanged("WLANVis");
}
}
public Visibility C2KVis
{
get
{
return c2kVis;
}
set
{
c2kVis = value;
OnPropertyChanged("C2KVis");
}
}
public Visibility EVDOVis
{
get
{
return evdoVis;
}
set
```

```
{
evdoVis = value;
OnPropertyChanged("EVDOVis");
}
}
public Visibility LTEDownlinkVis
{
get
{
return lteDownlinkVis;
}
set
{
lteDownlinkVis = value;
OnPropertyChanged("LTEDownlinkVis");
}
}

private bool metadataEditingEnabled;
public bool MetadataEditingEnabled
{
get
{
return metadataEditingEnabled;
}
set
{
metadataEditingEnabled = value;
OnPropertyChanged("MetadataEditingEnabled");
if(value == true)
{
OnChangeOccured(); //If meta data is enabled then assume changes are being made.
}
}
}

public ObservableCollection<KeyValuePair<Format?, string>> ComboboxRadioFormatList { get;
set; }
public ObservableCollection<KeyValuePair<RadioConfig?, string>> ComboboxRadioConfigList {
get; set; }
public ObservableCollection<KeyValuePair<LTEDownlinkMode?, string>>
ComboboxLTEDownlinkModeList { get; set; }
```

```
public ObservableCollection<double?> ComboboxSubcarrierSpacingList { get; set; }
public ObservableCollection<KeyValuePair<CyclicPrefix?, string>> ComboboxCyclicPrefixList {
get; set; }
public ObservableCollection<bool?> ComboboxTransformPrecodingList { get; set; }
public ObservableCollection<KeyValuePair<PUSCHModulation?, string>>
ComboboxPUSCHModList { get; set; }
public ObservableCollection<double?> ComboboxGuardIntervalList { get; set; }
public ObservableCollection<int?> ComboboxMCSList { get; set; }
public ObservableCollection<KeyValuePair<DataRate?, string>> ComboboxDataRateList { get;
set; }
public ObservableCollection<KeyValuePair<LTFCompType?, string>>
ComboboxLTFCompTypeList { get; set; }
public ObservableCollection<KeyValuePair<PhysicalLayerSubtype?, string>>
ComboboxPhysLayerSubtypeList { get; set; }


public KeyValuePair<Format?, string> ComboboxRadioFormatSelectedValue
{
get
{
return ComboboxRadioFormatList.Single(pair => pair.Key == this.RadioFormat);
}
set
{
this.RadioFormat = value.Key;
OnPropertyChanged("ComboboxRadioFormatSelectedValue");
}
}
public KeyValuePair<RadioConfig?, string> ComboboxRadioConfigSelectedValue
{
get
{
return ComboboxRadioConfigList.Single(pair => pair.Key == this.RConfig);
}
set
{
this.RConfig = value.Key;
OnPropertyChanged("ComboboxRadioConfigSelectedValue");
}
}
public KeyValuePair<LTEDownlinkMode?, string> ComboboxLTEDownlinkModeSelectedValue
{
```

```
get
{
return ComboboxLTEDownlinkModeList.Single(pair => pair.Key == this.DLMode);
}
set
{
this.DLMode = value.Key;
OnPropertyChanged("ComboboxLTEDownlinkModeSelectedValue");
}
}
public double? ComboboxSubcarrierSpacingSelectedValue
{
get
{
return ComboboxSubcarrierSpacingList.Single(a => a == this.SubcarrierSpacing);
}
set
{
this.SubcarrierSpacing = value;
OnPropertyChanged("ComboboxSubcarrierSpacingSelectedValue");
}
}
public KeyValuePair<CyclicPrefix?, string> ComboboxCyclicPrefixSelectedValue
{
get
{
return ComboboxCyclicPrefixList.Single(pair => pair.Key == this.CycPrefix);
}
set
{
this.CycPrefix = value.Key;
OnPropertyChanged("ComboboxCyclicPrefixSelectedValue");
}
}
public bool? ComboboxTransformPrecodingSelectedValue
{
get
{
return ComboboxTransformPrecodingList.Single(a => a == this.TransformPrecoding);
}
set
{
```

```
this.TransformPrecoding = value;
OnPropertyChanged("ComboboxTransformPrecodingSelectedValue");
}
}

public KeyValuePair<PUSCHModulation?, string> ComboboxPUSCHModSelectedValue
{
get
{
return ComboboxPUSCHModList.Single(pair => pair.Key == this.PUSCHMod);
}
set
{
this.PUSCHMod = value.Key;
OnPropertyChanged("ComboboxPUSCHModSelectedValue");
}
}
public double? ComboboxGuardIntervalSelectedValue
{
get
{
return ComboboxGuardIntervalList.Single(a => a == this.GuardInterval);
}
set
{
this.GuardInterval = value;
OnPropertyChanged("ComboboxGuardIntervalSelectedValue");
}
}
public int? ComboboxMCSSelectedValue
{
get
{
return ComboboxMCSList.Single(a => a == this.MCS);
}
set
{
this.MCS = value;
OnPropertyChanged("ComboboxMCSSelectedValue");
}
}
public KeyValuePair<DataRate?, string> ComboboxDataRateSelectedValue
```

```
{
get
{
return ComboboxDataRateList.Single(pair => pair.Key == this.DRate);
}
set
{
this.DRate = value.Key;
OnPropertyChanged("ComboboxDataRateSelectedValue");
}
}
public KeyValuePair<LTFCompType?, string> ComboboxLTFCompTypeSelectedValue
{
get
{
return ComboboxLTFCompTypeList.Single(pair => pair.Key == this.LTFType);
}
set
{
this.LTFType = value.Key;
OnPropertyChanged("ComboboxLTFCompTypeSelectedValue");
}
}
public KeyValuePair<PhysicalLayerSubtype?, string> ComboboxPhysLayerSubtypeSelectedValue
{
get
{
return ComboboxPhysLayerSubtypeList.Single(pair => pair.Key == this.PhysLayerSubtype);
}
set
{
this.PhysLayerSubtype = value.Key;
OnPropertyChanged("ComboboxPhysLayerSubtypeSelectedValue");
}
}


/// <summary>
/// Gets / Sets the object that is bound to the ListBox.
/// </summary>
public ObservableCollection<string> LoggingCollection { get; set; } = new
ObservableCollection<string>();
```

```
/// <summary>
/// Waveform crest factor
/// </summary>
public float? CrestFactor
{
get
{
return crestFactor;
}
set
{
crestFactor = value;
OnPropertyChanged("CrestFactor");
}
}

/// <summary>
/// Waveform Duty Cycle.
/// </summary>
public float? DutyCycle
{
get
{
return dutyCycle;
}
set
{
dutyCycle = value;
OnPropertyChanged("DutyCycle");
}
}

/// <summary>
/// Waveform ALC bandwidth
/// </summary>
public int? ALCBandwidth
{
get
{
return alcBandwidth;
}
```

```
set
{
alcBandwidth = value;
OnPropertyChanged("ALCBandwidth");
}
}

/// <summary>
/// Waveform number of samples
/// </summary>
public int? NumberOfSamples
{
get
{
return numberOfSamples;
}
set
{
numberOfSamples = value;
OnPropertyChanged("NumberOfSamples");
}
}

/// <summary>
/// Total time of waveform in ms.
/// </summary>
public double? TotalTime
{
get
{
return totalTime;
}
set
{
totalTime = value;
OnPropertyChanged("TotalTime");
}
}

/// <summary>
/// Waveform relative RMS.
/// </summary>
```

```
public float? RelRms
{
get
{
return relRms;
}
set
{
relRms = value;
OnPropertyChanged("RelRms");
}
}

/// <summary>
/// Waveform Sampling Rate in MHz
/// </summary>
public double? SamplingRate
{
get
{
return samplingRate;
}
set
{
samplingRate = value;
OnPropertyChanged("SamplingRate");
}
}

/// <summary>
/// Waveform RMS.
/// </summary>
public int? Rms
{
get
{
return rms;
}
set
{
rms = value;
OnPropertyChanged("Rms");
```

```
}
}

/// <summary>
/// Waveform Signal Bandwidth in MHz
/// </summary>
public double? SignalBandwidth
{
get
{
return signalBandwidth;
}
set
{
signalBandwidth = value;
OnPropertyChanged("SignalBandwidth");
}
}

/// <summary>
/// Waveform Description.
/// </summary>
public string Description
{
get
{
return description;
}
set
{
description = value;
OnPropertyChanged("Description");
}
}

/// <summary>
/// Waveform Radio Format.
/// </summary>
public Format? RadioFormat
{
get
{
```

```
return radioFormat;
}
set
{
radioFormat = value;
switch (radioFormat)
{
case Format.NR:
{
this.NRVis = Visibility.Visible;
this.WLANVis = Visibility.Collapsed;
this.C2KVis = Visibility.Collapsed;
this.EVDOVis = Visibility.Collapsed;
this.LTEDownlinkVis = Visibility.Collapsed;
break;
}
case Format.WLAN_AC:
case Format.WLAN_AG:
case Format.WLAN_AX:
case Format.WLAN_B:
case Format.WLAN_BE:
case Format.WLAN_J:
case Format.WLAN_N:
case Format.WLAN_P:
{
this.NRVis = Visibility.Collapsed;
this.WLANVis = Visibility.Visible;
this.C2KVis = Visibility.Collapsed;
this.EVDOVis = Visibility.Collapsed;
this.LTEDownlinkVis = Visibility.Collapsed;
break;
}
case Format.C2K:
{
this.NRVis = Visibility.Collapsed;
this.WLANVis = Visibility.Collapsed;
this.C2KVis = Visibility.Visible;
this.EVDOVis = Visibility.Collapsed;
this.LTEDownlinkVis = Visibility.Collapsed;
break;
}
case Format._1xEvDo:
```

```
{
this.NRVis = Visibility.Collapsed;
this.WLANVis = Visibility.Collapsed;
this.C2KVis = Visibility.Collapsed;
this.EVDOVis = Visibility.Visible;
this.LTEDownlinkVis = Visibility.Collapsed;
break;
}
case Format.LTE_Downlink:
{
this.NRVis = Visibility.Collapsed;
this.WLANVis = Visibility.Collapsed;
this.C2KVis = Visibility.Collapsed;
this.EVDOVis = Visibility.Collapsed;
this.LTEDownlinkVis = Visibility.Visible;
break;
}
default:
{
this.NRVis = Visibility.Collapsed;
this.WLANVis = Visibility.Collapsed;
this.C2KVis = Visibility.Collapsed;
this.EVDOVis = Visibility.Collapsed;
this.LTEDownlinkVis = Visibility.Collapsed;
break;
}
}
OnPropertyChanged("RadioFormat");
}
}

/// <summary>
/// Measurement offset in ms.
/// </summary>
public double? Offset
{
get
{
return offset;
}
set
{
```

```
offset = value;
OnPropertyChanged("Offset");
}
}

/// <summary>
/// Measurement length in ms.
/// </summary>
public double? MeasLength
{
get
{
return measLength;
}
set
{
measLength = value;
OnPropertyChanged("MeasLength");
}
}

/// <summary>
/// Step length in ms.
/// </summary>
public double? StepLength
{
get
{
return stepLength;
}
set
{
stepLength = value;
OnPropertyChanged("StepLength");
}
}

/// <summary>
/// Number of Steps.
/// </summary>
public int? NumSteps
{
```

```
get
{
return numSteps;
}
set
{
numSteps = value;
OnPropertyChanged("NumSteps");
}
}

/// <summary>
/// Subcarrier Spacing in kHz
/// </summary>
public double? SubcarrierSpacing
{
get
{
return subcarrierSpacing;
}
set
{
subcarrierSpacing = value;
OnPropertyChanged("SubcarrierSpacing");
}
}

/// <summary>
/// Cyclic Prefix
/// </summary>
public CyclicPrefix? CycPrefix
{
get
{
return cycPrefix;
}
set
{
cycPrefix = value;
OnPropertyChanged("CycPrefix");
}
}
```

```
/// <summary>
/// Transform precoding, DFT-s-OFDM = true, CP-OFDM = false
/// </summary>
public bool? TransformPrecoding
{
get
{
return transformPrecoding;
}
set
{
transformPrecoding = value;
OnPropertyChanged("TransformPrecoding");
}
}

/// <summary>
/// PUSCH modulation
/// </summary>
public PUSCHModulation? PUSCHMod
{
get
{
return puschMod;
}
set
{
puschMod = value;
OnPropertyChanged("PUSCHMod");
}
}

/// <summary>
/// PUSCH Slots
/// </summary>
public string PUSCHSlots
{
get
{
return puschSlots;
}
```

```
set
{
puschSlots = value;
OnPropertyChanged("PUSCHSlots");
}
}

/// <summary>
/// PUSCH Physical Resource Blocks
/// </summary>
public string PUSCHPRB
{
get
{
return puschPRB;
}
set
{
puschPRB = value;
OnPropertyChanged("PUSCHPRB");
}
}

/// <summary>
/// Guard Interval in us
/// </summary>
public double? GuardInterval
{
get
{
return guardInterval;
}
set
{
guardInterval = value;
OnPropertyChanged("GuardInterval");
}
}

/// <summary>
/// Modulation Coding Scheme
/// </summary>
```

```
public int? MCS
{
get
{
return mcs;
}
set
{
mcs = value;
OnPropertyChanged("MCS");
}
}

/// <summary>
/// Data Rate (802.11b)
/// </summary>
public DataRate? DRate
{
get
{
return dRate;
}
set
{
dRate = value;
OnPropertyChanged("DRate");
}
}

/// <summary>
/// Long Training Field Compression Type (802.11ax/be)
/// </summary>
public LTFCompType? LTFType
{
get
{
return ltfType;
}
set
{
ltfType = value;
OnPropertyChanged("LTFType");
```

```
}
}

/// <summary>
/// Number of data OFDM symbols
/// </summary>
public int? DataSymbols
{
get
{
return dataSymbols;
}
set
{
dataSymbols = value;
OnPropertyChanged("DataSymbols");
}
}

/// <summary>
/// Radio Config (C2K)
/// </summary>
public RadioConfig? RConfig
{
get
{
return rConfig;
}
set
{
rConfig = value;
OnPropertyChanged("RConfig");
}
}

/// <summary>
/// Physical Layer Subtype 1XEVDO
/// </summary>
public PhysicalLayerSubtype? PhysLayerSubtype
{
get
{
```

```
return physLayerSubtype;
}
set
{
physLayerSubtype = value;
OnPropertyChanged("PhysLayerSubtype");
}
}

/// <summary>
/// LTE Downlink Mode
/// </summary>
public LTEDownlinkMode? DLMode
{
get
{
return dlMode;
}
set
{
dlMode = value;
OnPropertyChanged("DLMode");
}
}

/// <summary>
/// LTE Config Index
/// </summary>
public int? ULDLConfigIndex
{
get
{
return uldlConfigIndex;
}
set
{
uldlConfigIndex = value;
OnPropertyChanged("ULDLConfigIndex");
}
}

/// <summary>
```

```
/// LTE physical downlink control channel num symbol
/// </summary>
public int? PDCCHNumSymbol
{
get
{
return pdcchNumSymbol;
}
set
{
pdcchNumSymbol = value;
OnPropertyChanged("PDCCHNumSymbol");
}
}

/// <summary>
/// Gets the title for the app with version number appended )
/// </summary>
public string AppTitle
{
get
{
return string.Format("Merlin Test Technologies - Waveform Visualizer");
}
}

public ObservableCollection<MarkerSegment> Marker1Segments { get; set; } = new
ObservableCollection<MarkerSegment>();
public ObservableCollection<MarkerSegment> Marker2Segments { get; set; } = new
ObservableCollection<MarkerSegment>();
public ObservableCollection<MarkerSegment> Marker3Segments { get; set; } = new
ObservableCollection<MarkerSegment>();
public ObservableCollection<MarkerSegment> Marker4Segments { get; set; } = new
ObservableCollection<MarkerSegment>();

public int TabSelectedIndex { get; set; } = -1;

public TabItem TabSelected { get; set; } = null;

#endregion Public Properties

#region Commands
```

```
#region Load AIQ File

/// <summary>
/// A method that can be invoked by Task.Run to load the .aiq file asynchronously.
/// </summary>
/// <param name="fileName"></param>
/// <returns></returns>
private async Task LoadAiqFile(string fileName)
{
await Task.Run(() => LoadAiqFileWorker(fileName));
}

/// <summary>
/// Loads the .aiq file and stores the meta data, IQ data and marker data in some member
variables.
/// </summary>
/// <param name="fileName"></param>
private void LoadAiqFileWorker(string fileName)
{
MetadataEditingEnabled = false; //triggers IsSaveRequired...
int iDataWithMarker;
int qDataWithMarker;
double powerInWatts;

// Clear previous files data.
Idata.Clear();
Qdata.Clear();
Power.Clear();
Marker1.Clear();
Marker2.Clear();
Marker3.Clear();
Marker4.Clear();

// Null out all waveform parameters
m_currentMTWaveform = null;
this.Description = null;
this.ALCBandwidth = null;

this.ComboboxRadioFormatSelectedValue = ComboboxRadioFormatList.Where(a => a.Key ==
null).FirstOrDefault();
this.SamplingRate = null;
```

```
this.RelRms = null;
this.Rms = null;
this.SignalBandwidth = null;
this.Offset = null;
this.MeasLength = null;
this.StepLength = null;
this.NumSteps = null;
this.DutyCycle = null;
this.CrestFactor = null;
this.TotalTime = null;
this.NumberOfSamples = null;

//NR
this.ComboboxSubcarrierSpacingSelectedValue = ComboboxSubcarrierSpacingList.Where(a => a
== null).FirstOrDefault();
this.ComboboxCyclicPrefixSelectedValue = ComboboxCyclicPrefixList.Where(a => a.Key ==
null).FirstOrDefault();
this.ComboboxTransformPrecodingSelectedValue = ComboboxTransformPrecodingList.Where(a
=> a == null).FirstOrDefault();
this.ComboboxPUSCHModSelectedValue = ComboboxPUSCHModList.Where(a => a.Key ==
null).FirstOrDefault();
this.PUSCHSlots = null;
this.PUSCHPRB = null;

//WLAN
this.ComboboxGuardIntervalSelectedValue = ComboboxGuardIntervalList.Where(a => a ==
null).FirstOrDefault();
this.ComboboxMCSSelectedValue = ComboboxMCSList.Where(a => a == null).FirstOrDefault();
this.ComboboxDataRateSelectedValue = ComboboxDataRateList.Where(a => a.Key ==
null).FirstOrDefault();
this.ComboboxLTFCompTypeSelectedValue = ComboboxLTFCompTypeList.Where(a => a.Key
== null).FirstOrDefault();
this.DataSymbols = null;

//C2K
this.ComboboxRadioConfigSelectedValue = ComboboxRadioConfigList.Where(a => a.Key ==
null).FirstOrDefault();

//1xEVDO
this.ComboboxPhysLayerSubtypeSelectedValue = ComboboxPhysLayerSubtypeList.Where(a =>
a.Key == null).FirstOrDefault();
```

```
//LTE Downlink
this.ComboboxLTEDownlinkModeSelectedValue = ComboboxLTEDownlinkModeList.Where(a =>
a.Key == null).FirstOrDefault();
this.ULDLConfigIndex = null;
this.PDCCHNumSymbol = null;

clearMarkerSegments();

// Create an array of the correct size to read in the IQ data with marker information.
int[] iqdatawithmarkers = new int[currentLoadedAiqFile.I.Length];

// Load in IQ data with marker information.
currentLoadedAiqFile.ReadIQWithMarkerData(0, currentLoadedAiqFile.I.Length,
iqdatawithmarkers);

short dataAndMarkerAsShort, dataAsShort;
// For each IQ Sample fetch out the I sample, Q Sample, Marker 1/2/3/4 information and
calculation power in dBm per sample.
for (int index = 0; index < iqdatawithmarkers.Length; index++)
{
// I Data and markers
iDataWithMarker = iqdatawithmarkers[index] & 0xffff;

// Cast to Int16 so the bitshit doesn't cause any issues with negative numbers.
dataAndMarkerAsShort = (Int16)(iDataWithMarker);

// Bit shift by 2 to get just the I-Data.
dataAsShort = (Int16)(dataAndMarkerAsShort >> 2);

//Idata.Add(((double)(iDataWithMarker >> 2))/8192);
Idata.Add(((double)(dataAsShort)) / 8192);

// Get the 1st and second bits out for markers 3 and 4.
Marker3.Add(iDataWithMarker & 0x1);
Marker4.Add(iDataWithMarker & 0x2);

// Q Data and markers
qDataWithMarker = iqdatawithmarkers[index] >> 16;

// Cast to Int16 so the bitshit doesn't cause any issues with negative numbers
dataAndMarkerAsShort = (Int16)(qDataWithMarker);
```

```
// Bit shift by 2 to get just the Q-Data.
dataAsShort = (Int16)(dataAndMarkerAsShort >> 2);

Qdata.Add(((double)(dataAsShort)) / 8192);

// Get the 1st and second bits out for markers 3 and 4.
Marker1.Add(qDataWithMarker & 0x1);
Marker2.Add(qDataWithMarker & 0x2);

// Calculate Power
powerInWatts = Idata[index] * Idata[index] + Qdata[index] * Qdata[index];

// Occasionaly we get IQ samples with a value of zero. To avoid an issue we just set the value
// to a really small value which avoid the issue when charting.
if (powerInWatts == 0D)
{
powerInWatts = 0.0000000000001;
}

// Store Result.
Power.Add(10.0 * Math.Log10(powerInWatts));
}

// Parse description for metadata if possible
this.Description = currentLoadedAiqFile.Description;
string[] metadataArray = this.Description.Split(',');
if (metadataArray[0] == "Ver") // new .aiq format begins with "Ver" in the description field
{
for (int i = 0; i < metadataArray.Length - 1; i += 2)
{
string field = metadataArray[i];
string value = metadataArray[i + 1];
switch (field)
{
case "Format":
if (Enum.TryParse(value, out Format format) == true)
{
this.ComboboxRadioFormatSelectedValue = ComboboxRadioFormatList.Where(a => a.Key ==
format).FirstOrDefault();
}
else
{
```

```
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "NSamples":
if (int.TryParse(value, out int nSamples) == true)
{
this.NumberOfSamples = nSamples;
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "SR":
if (double.TryParse(value, out double sr) == true)
{
this.SamplingRate = sr; //MHz
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "RelRMS":
if (float.TryParse(value, out float relrms) == true)
{
this.RelRms = relrms;
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "BW":
if (double.TryParse(value, out double bw) == true)
{
this.SignalBandwidth = bw; //MHz
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
```

```
break;
case "Offset":
if (double.TryParse(value, out double offset) == true)
{
this.Offset = offset; //ms
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "MeasLen":
if (double.TryParse(value, out double measlen) == true)
{
this.MeasLength = measlen; //ms
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "StepLen":
if (double.TryParse(value, out double steplen) == true)
{
this.StepLength = steplen; //ms
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "NumSteps":
if (int.TryParse(value, out int numsteps) == true)
{
this.NumSteps = numsteps;
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "DC":
```

```
if (float.TryParse(value, out float dc) == true)
{
this.DutyCycle = dc;
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "CF":
if (float.TryParse(value, out float cf) == true)
{
this.CrestFactor = cf;
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "WFLen":
if (double.TryParse(value, out double wflen) == true)
{
this.TotalTime = wflen; //ms
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "SCS":
if (double.TryParse(value, out double scs) == true)
{
this.ComboboxSubcarrierSpacingSelectedValue = ComboboxSubcarrierSpacingList.Where(a => a
== scs).FirstOrDefault(); //kHz
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "CP":
if (Enum.TryParse(value, out CyclicPrefix cp) == true)
```

```
{
this.ComboboxCyclicPrefixSelectedValue = ComboboxCyclicPrefixList.Where(a => a.Key ==
cp).FirstOrDefault();
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "TPrecoding":
if (bool.TryParse(value, out bool tprecoding) == true)
{
this.ComboboxTransformPrecodingSelectedValue = ComboboxTransformPrecodingList.Where(a
=> a == tprecoding).FirstOrDefault();
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "PUSCHMod":
if (Enum.TryParse(value, out PUSCHModulation puschmod) == true)
{
this.ComboboxPUSCHModSelectedValue = ComboboxPUSCHModList.Where(a => a.Key ==
puschmod).FirstOrDefault();
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "PUSCHSlots":
this.PUSCHSlots = value;
break;
case "PUSCHPRB":
this.PUSCHPRB = value;
break;
case "GI":
if (double.TryParse(value, out double gi) == true)
{
this.ComboboxGuardIntervalSelectedValue = ComboboxGuardIntervalList.Where(a => a ==
gi).FirstOrDefault(); //us
```

```
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "MCS":
if (value == "N/A")
{
break;
}
else
{
if (int.TryParse(value, out int mcs) == true)
{
this.ComboboxMCSSelectedValue = ComboboxMCSList.Where(a => a == mcs).FirstOrDefault();
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
}
case "DataRate":
if (value == "N/A")
{
break;
}
else
{
if (Enum.TryParse(value, out DataRate dr) == true)
{
this.ComboboxDataRateSelectedValue = ComboboxDataRateList.Where(a => a.Key ==
dr).FirstOrDefault();
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
}
case "LTFType":
```

```
if (value == "N/A")
{
break;
}
else
{
if (Enum.TryParse(value, out LTFCompType ltfct) == true)
{
this.ComboboxLTFCompTypeSelectedValue = ComboboxLTFCompTypeList.Where(a => a.Key
== ltfct).FirstOrDefault();
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
}
case "DataSymbols":
if (value == "N/A")
{
break;
}
else
{
if (int.TryParse(value, out int ds) == true)
{
this.DataSymbols = ds;
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
}
case "RConfig":
if (Enum.TryParse(value, out RadioConfig rc) == true)
{
this.ComboboxRadioConfigSelectedValue = ComboboxRadioConfigList.Where(a => a.Key ==
rc).FirstOrDefault();
}
else
{
```

```
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "PhysLayerSubtype":
if (Enum.TryParse(value, out PhysicalLayerSubtype plst) == true)
{
this.ComboboxPhysLayerSubtypeSelectedValue = ComboboxPhysLayerSubtypeList.Where(a =>
a.Key == plst).FirstOrDefault();
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "DLMode":
if (Enum.TryParse(value, out LTEDownlinkMode dlm) == true)
{
this.ComboboxLTEDownlinkModeSelectedValue = ComboboxLTEDownlinkModeList.Where(a =>
a.Key == dlm).FirstOrDefault();
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "ULDLConfigIndex":
if (int.TryParse(value, out int uldlci) == true)
{
this.ULDLConfigIndex = uldlci;
}
else
{
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
case "PDCCHNumSymbol":
if (int.TryParse(value, out int pddchns) == true)
{
this.PDCCHNumSymbol = pddchns;
}
else
{
```

```
throw new FormatException("Unrecognized metadata type in .aiq file description field");
}
break;
}
}
}
else // old format
{
//Calculate DC, CF, and RelRMS. The CF and RelRMS values embedded in the legacy aiq files are not what we want
float calcRMS;
float calcCF;
float calcDC;
List<double> modifiedIDataWithoutZeros = new List<double>();
List<double> modifiedQDataWithoutZeros = new List<double>();
List<double> sumOfSq = new List<double>();

// Removing zero values
for (int index = 0; index < this.Idata.Count; index++)
{
if (Math.Sqrt(this.Idata[index] * this.Idata[index] + this.Qdata[index] + this.Qdata[index]) != 0)
{
modifiedIDataWithoutZeros.Add(this.Idata[index]);
modifiedQDataWithoutZeros.Add(this.Qdata[index]);
}
}

// Calculate DC
calcDC = ((float)modifiedIDataWithoutZeros.Count / (float)this.Idata.Count) * 100;

// Calculate Power per sample of the list without zeros
for (int index = 0; index < modifiedIDataWithoutZeros.Count; index++)
{
sumOfSq.Add(modifiedIDataWithoutZeros[index] * modifiedIDataWithoutZeros[index] +
modifiedQDataWithoutZeros[index] * modifiedQDataWithoutZeros[index]);
}

// Take the average power and take the sqrt to get RMS ( as calculated by Keysight )
calcRMS = (float)Math.Sqrt(sumOfSq.Average());

// Now calculate CF, keeping in mind the peak is 1
calcCF = (float)(20 * Math.Log10(1 / calcRMS));
```

```
// Populate metadata
this.SignalBandwidth = currentLoadedAiqFile.SignalBandwidth / 1e6; //MHz
this.CrestFactor = calcCF;
this.ALCBandwidth = currentLoadedAiqFile.AlcBandwidth; //Load this value but it's significance is unknown. Not used anywhere in this GUI
this.NumberOfSamples = currentLoadedAiqFile.NumberOfSamples;
this.RelRms = calcRMS;
this.Rms = currentLoadedAiqFile.Rms; //Load this value but it's significance is unknown. Not used anywhere in this GUI
this.SamplingRate = currentLoadedAiqFile.SamplingRate / 1e6; //MHz
this.TotalTime = (this.NumberOfSamples / this.SamplingRate) / 1000; // Convert to ms
this.DutyCycle = calcDC;
}

MarkerSegment mk1Seg = null;
MarkerSegment mk2Seg = null;
MarkerSegment mk3Seg = null;
MarkerSegment mk4Seg = null;

for (int index = 0; index < Idata.Count; index++)
{
determineMarkerSegment(Marker1[index], index, Idata.Count - 1, this.Marker1Segments, ref mk1Seg);
determineMarkerSegment(Marker2[index], index, Idata.Count - 1, this.Marker2Segments, ref mk2Seg);
determineMarkerSegment(Marker3[index], index, Idata.Count - 1, this.Marker3Segments, ref mk3Seg);
determineMarkerSegment(Marker4[index], index, Idata.Count - 1, this.Marker4Segments, ref mk4Seg);
}
}

private void clearMarkerSegments()
{
this.Marker1Segments.Clear();
this.Marker2Segments.Clear();
this.Marker3Segments.Clear();
this.Marker4Segments.Clear();
}

private void determineMarkerSegment(int markerValue, int markerIndex, int maxIndex,
```

```
ObservableCollection<MarkerSegment> segments, ref MarkerSegment markerSegment)
{
if (markerValue != 0)
{
if (markerSegment == null)
markerSegment = new MarkerSegment() { From = markerIndex + 1, To = markerIndex + 1 };
else
markerSegment.To = markerIndex + 1;

if (markerIndex == maxIndex)//achieve the end
{
segments.Add(markerSegment);
markerSegment = null;
}
}
else
{
if (markerSegment != null)
{
segments.Add(markerSegment);
markerSegment = null;
}
}
}

/// <summary>
/// Loads in a .aiq waveform file.
/// </summary>
public string OpenAiqFileCommandExecute(string optFilename = null)
{
try
{
if (String.IsNullOrEmpty(optFilename))
{
// Configure open file dialog box
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();
dlg.DefaultExt = ".aiq"; // Default file extension
dlg.Filter = "Aeroflex Waveform File (.aiq)|*.aiq"; // Filter files by extension
// Show open file dialog box
Nullable<bool> result = dlg.ShowDialog();
if (result == true)
{
```

```
this.InputFileName = dlg.FileName;
using (currentLoadedAiqFile = new AiqFileReader(dlg.FileName))
{
LoadAiqFileWorker(dlg.FileName);
}
View.SetupChart(null, "Samples", "Time (ms)", "Power (dBm)", 0, 0,
(double)this.NumberOfSamples, ((double)this.NumberOfSamples / (double)this.SamplingRate) /
1e3);
drawWholeChart();
return String.Format("Successfully loaded {0}", dlg.FileName);
}
else
{
return null;
}
}
else
{
this.InputFileName = optFilename;
using (currentLoadedAiqFile = new AiqFileReader(optFilename))
{
LoadAiqFileWorker(optFilename);
}
View.SetupChart(null, "Samples", "Time (ms)", "Power (dBm)", 0, 0,
(double)this.NumberOfSamples, ((double)this.NumberOfSamples / (double)this.SamplingRate) /
1e3);
drawWholeChart();
return String.Format("Successfully loaded {0}", optFilename);
}
}
catch (Exception ex)
{
MessageBox.Show("An error occurred while loading .aiq file: " + ex.Message, "Merlin Test Studio",
MessageBoxButton.OK, MessageBoxImage.Error);
return null;
}
}

#endregion Load AIQ File

#region Save MTWF
```

```
/// <summary>
/// Saves loaded .aiq file into the Merlin Test waveform format and the Keysight X-APP BIN format.
/// </summary>
public string SaveAsMTWFMethod()
{
try
{
//string defaultPath = @"C:\ProgramData\Merlin Test\Data Analysis\";
// Configure open file dialog box
Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();
dlg.FileName = Path.GetFileNameWithoutExtension(this.InputFileName);
dlg.DefaultExt = ".mtwf2"; // Default file extension
dlg.Filter = "Merlin Test Waveform File (.mtwf2)|*.mtwf2"; // Filter files by extension
dlg.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) +
"\\MerlinTest";

// Show save file dialog box
Nullable<bool> result = dlg.ShowDialog();

// Process save file dialog box results
if (result == true)
{
return SaveToMTWF(dlg.FileName);
}
else
{
return null;
}
}
catch (Exception ex)
{
MessageBox.Show("An error occurred while saving .mtwf2 file: " + ex.Message, "Merlin Test
Studio", MessageBoxButton.OK, MessageBoxImage.Error);
return null;
}
}
private string SaveToMTWF(string mtwfFullFileName)
{
//Create new MT waveform object
MTWaveform mtwfObj = new MTWaveform();

//Populate metadata based on previous load or any user edits
```

```
EmbedMetadataInMTWFObj(mtwfObj);

//Populate IQ and marker data
for (int i=0; i<this.Idata.Count; i++)
{
mtwfObj.IData.Add(Convert.ToSingle(this.Idata[i]));
mtwfObj.QData.Add(Convert.ToSingle(this.Qdata[i]));
byte finalValue = 0x0;

if (this.Marker1[i] != 0)
{
finalValue = (byte)(finalValue | 0x1);
}
if (this.Marker2[i] != 0)
{
finalValue = (byte)(finalValue | 0x2);
}
if (this.Marker3[i] != 0)
{
finalValue = (byte)(finalValue | 0x4);
}
if (this.Marker4[i] != 0)
{
finalValue = (byte)(finalValue | 0x8);
}
mtwfObj.Marker.Add(finalValue);
}

mtwfObj.SaveToBinary(mtwfFullFileName);
MetadataEditingEnabled = false;
return String.Format("Successfully saved {0}", mtwfFullFileName);
}
private void EmbedMetadataInMTWFObj(MTWaveform mtWaveform)
{
// Embed metadata from the ViewModel, whether it's there from the previous file load or the user has added/overwritten it
if (this.ComboboxRadioFormatSelectedValue.Key != null)
{
mtWaveform.Format = (Format)this.ComboboxRadioFormatSelectedValue.Key;
}
if (this.SamplingRate != null)
{
```

```
mtWaveform.SR = (double)this.SamplingRate * 1e6;
}
if (this.NumberOfSamples != null)
{
mtWaveform.NSamples = (double)this.NumberOfSamples;
}
if (this.DutyCycle != null)
{
mtWaveform.DC = (double)this.DutyCycle;
}
if (this.CrestFactor != null)
{
mtWaveform.CF = (double)this.CrestFactor;
}
if (this.SignalBandwidth != null)
{
mtWaveform.BW = (double)this.SignalBandwidth * 1e6;
}
if (this.RelRms != null)
{
mtWaveform.RelRMS = (double)this.RelRms;
}
if (this.TotalTime != null)
{
mtWaveform.WFLen = (double)this.TotalTime / 1e3;
}
if (this.Offset != null)
{
mtWaveform.Offset = (double)this.Offset / 1e3;
}
if (this.MeasLength != null)
{
mtWaveform.MeasLen = (double)this.MeasLength / 1e3;
}
if (this.StepLength != null)
{
mtWaveform.StepLen = (double)this.StepLength / 1e3;
}
if (this.NumSteps != null)
{
mtWaveform.NumSteps = (int)this.NumSteps;
}
```

```
if (this.ComboboxSubcarrierSpacingSelectedValue != null)
{
mtWaveform.SCS = (double)this.ComboboxSubcarrierSpacingSelectedValue * 1e3;
}
if (this.ComboboxCyclicPrefixSelectedValue.Key != null)
{
mtWaveform.CP = (CyclicPrefix)this.ComboboxCyclicPrefixSelectedValue.Key;
}
if (this.ComboboxTransformPrecodingSelectedValue != null)
{
mtWaveform.TPrecoding = (bool)this.ComboboxTransformPrecodingSelectedValue;
}
if (this.ComboboxPUSCHModSelectedValue.Key != null)
{
mtWaveform.PUSCHMod = (PUSCHModulation)this.ComboboxPUSCHModSelectedValue.Key;
}
if (this.PUSCHSlots != null)
{
mtWaveform.PUSCHSlots = this.PUSCHSlots;
}
if (this.PUSCHPRB != null)
{
mtWaveform.PUSCHPRB = this.PUSCHPRB;
}
if (this.ComboboxGuardIntervalSelectedValue != null)
{
mtWaveform.GI = (double)this.ComboboxGuardIntervalSelectedValue / 1e6;
}
if (this.ComboboxMCSSelectedValue != null)
{
mtWaveform.MCS = (int)this.ComboboxMCSSelectedValue;
}
if (this.ComboboxDataRateSelectedValue.Key != null)
{
mtWaveform.DataRate = (DataRate)this.ComboboxDataRateSelectedValue.Key;
}
if (this.ComboboxLTFCompTypeSelectedValue.Key != null)
{
mtWaveform.LTFType = (LTFCompType)this.ComboboxLTFCompTypeSelectedValue.Key;
}
if (this.DataSymbols != null)
{
```

```
mtWaveform.DataSymbols = (int)this.DataSymbols;
}
if (this.ComboboxRadioConfigSelectedValue.Key != null)
{
mtWaveform.RConfig = (RadioConfig)this.ComboboxRadioConfigSelectedValue.Key;
}
if (this.ComboboxPhysLayerSubtypeSelectedValue.Key != null)
{
mtWaveform.PLSubtype =
(PhysicalLayerSubtype)this.ComboboxPhysLayerSubtypeSelectedValue.Key;
}
if (this.ComboboxLTEDownlinkModeSelectedValue.Key != null)
{
mtWaveform.DLMode =
(LTEDownlinkMode)this.ComboboxLTEDownlinkModeSelectedValue.Key;
}
if (this.ULDLConfigIndex != null)
{
mtWaveform.ConfigInd = (int)this.ULDLConfigIndex;
}
if (this.PDCCHNumSymbol != null)
{
mtWaveform.PdcchNumSymbol = (int)this.PDCCHNumSymbol;
}
}

#endregion Save MTWF

#region Save AIQ
/// <summary>
/// Saves loaded .aiq file in native (.aiq) format.
/// </summary>
public string SaveAsAIQMethod()
{
try
{
// Configure open file dialog box
Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();
dlg.FileName = Path.GetFileNameWithoutExtension(this.InputFileName);
dlg.DefaultExt = ".aiq"; // Default file extension
dlg.Filter = "Aeroflex Waveform File (.aiq)|*.aiq"; // Filter files by extension
dlg.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) +
```

```
"\\MerlinTest";

// Show save file dialog box
Nullable<bool> result = dlg.ShowDialog();

// Process save file dialog box results
if (result == true)
{
return SaveToAIQ(dlg.FileName);
}
else
{
return null;
}
}
catch (Exception ex)
{
MessageBox.Show("An error occurred while saving .aiq file: " + ex.Message, "Merlin Test Studio",
MessageBoxButton.OK, MessageBoxImage.Error);
return null;
}
}
private string SaveToAIQ(string aiqFullFileName)
{
// Write IQ data to .txt file for input to aiq packager .dll
string txtFileName = Path.ChangeExtension(aiqFullFileName, ".txt");
using (StreamWriter sw = new StreamWriter(File.Open(txtFileName, FileMode.Create)))
{
for (int index = 0; index < this.Idata.Count; index++)
{
sw.WriteLine(this.Idata[index]);
sw.WriteLine(this.Qdata[index]);
}
}

// Write marker data to .mkr file for input to aiq packager dll
string mkrFileName = Path.ChangeExtension(aiqFullFileName, ".mkr");
using (StreamWriter sw = new StreamWriter(File.Open(mkrFileName, FileMode.Create)))
{
sw.WriteLine("FrameLength={0}",this.Idata.Count);
sw.WriteLine("Oversampling=1");
sw.WriteLine("Decimation=1");
```

```
sw.WriteLine("Mkr1=General");
if(Marker1Segments.Count!=0)
{
sw.WriteLine("Marker 1");
for (int idx = 0; idx < Marker1Segments.Count; idx++)
{
sw.WriteLine("{0},{1}", Marker1Segments[idx].From-1, Marker1Segments[idx].To);
}

}
sw.WriteLine("Mkr2=General");
if (Marker2Segments.Count != 0)
{
sw.WriteLine("Marker 2");
for (int idx = 0; idx < Marker2Segments.Count; idx++)
{
sw.WriteLine("{0},{1}", Marker2Segments[idx].From-1, Marker2Segments[idx].To);
}

}
sw.WriteLine("Mkr3=General");
if (Marker3Segments.Count != 0)
{
sw.WriteLine("Marker 3");
for (int idx = 0; idx < Marker3Segments.Count; idx++)
{
sw.WriteLine("{0},{1}", Marker3Segments[idx].From-1, Marker3Segments[idx].To);
}

}
sw.WriteLine("Mkr4=General");
if (Marker4Segments.Count != 0)
{
sw.WriteLine("Marker 4");
for (int idx = 0; idx < Marker4Segments.Count; idx++)
{
sw.WriteLine("{0},{1}", Marker4Segments[idx].From-1, Marker4Segments[idx].To);
}
}
}
```

```
// Create metadata string
string description = CreateAIQMetadataString();

// Define output filename
string outputFileName = Path.ChangeExtension(aiqFullFileName, ".aiq");

float sampleRate = (float)(this.SamplingRate * 1e6); //Hz

if(sampleRate > 200e6)
{
throw new Exception(".aiq files cannot be generated for sample rates exceeding 200 MHz.");
}

// Call package manager with source file w/IQ data, markers, description, dest filename, sample rate?
int errorCode = PackageAiqWaveform(txtFileName, mkrFileName, outputFileName, description, sampleRate);
MetadataEditingEnabled = false;
if (errorCode==0)
{
// Clean up .mkr and .txt files
File.Delete(mkrFileName);
File.Delete(txtFileName);
return String.Format("Successfully saved {0}", aiqFullFileName);
}
else
{
File.Delete(mkrFileName);
File.Delete(txtFileName);
throw new Exception("PackageAiqWaveform Error: " + errorCode.ToString());
}
}
private string CreateAIQMetadataString()
{
// Include version in description so it's parsed by viewer on subsequent loads
string description = "Ver,1";

// Embed metadata from the ViewModel, whether it's there from the previous file load or the user has added/overwritten it
if (this.ComboboxRadioFormatSelectedValue.Key != null &&
this.ComboboxRadioFormatSelectedValue.Key != Format.Undefined)
{
```

```
description += ",Format," + ((Format)this.ComboboxRadioFormatSelectedValue.Key).ToString();
}
if (this.SamplingRate != null)
{
description += ",SR," + ((double)this.SamplingRate).ToString(); //MHz
}
if (this.NumberOfSamples != null)
{
description += ",NSamples," + ((double)this.NumberOfSamples).ToString();
}
if (this.DutyCycle != null)
{
description += ",DC," + ((double)this.DutyCycle).ToString("0.###");
}
if (this.CrestFactor != null)
{
description += ",CF," + ((double)this.CrestFactor).ToString("0.###");
}
if (this.SignalBandwidth != null)
{
description += ",BW," + ((double)this.SignalBandwidth).ToString(); //MHz
}
if (this.RelRms != null)
{
description += ",RelRMS," + ((double)this.RelRms).ToString("0.###");
}
if (this.TotalTime != null)
{
description += ",WFLen," + ((double)this.TotalTime).ToString(); //ms
}
if (this.Offset != null)
{
description += ",Offset," + ((double)this.Offset).ToString(); //ms
}
if (this.MeasLength != null)
{
description += ",MeasLen," + ((double)this.MeasLength).ToString(); //ms
}
if (this.StepLength != null)
{
description += ",StepLen," + ((double)this.StepLength).ToString(); //ms
}
```

```
if (this.NumSteps != null)
{
description += ",NumSteps," + ((int)this.NumSteps).ToString();
}

switch(this.ComboboxRadioFormatSelectedValue.Key)
{
case Format.NR:
if (this.ComboboxSubcarrierSpacingSelectedValue != null)
{
description += ",SCS," + ((double)this.ComboboxSubcarrierSpacingSelectedValue).ToString();
//kHz
}
if (this.ComboboxCyclicPrefixSelectedValue.Key != null &&
this.ComboboxCyclicPrefixSelectedValue.Key != CyclicPrefix.Undefined)
{
description += ",CP," + ((CyclicPrefix)this.ComboboxCyclicPrefixSelectedValue.Key).ToString();
}
if (this.ComboboxTransformPrecodingSelectedValue != null)
{
description += ",TPrecoding," +
((bool)this.ComboboxTransformPrecodingSelectedValue).ToString();
}
if (this.ComboboxPUSCHModSelectedValue.Key != null &&
this.ComboboxPUSCHModSelectedValue.Key != PUSCHModulation.Undefined)
{
description += ",PUSCHMod," +
((PUSCHModulation)this.ComboboxPUSCHModSelectedValue.Key).ToString();
}
if (this.PUSCHSlots != null)
{
description += ",PUSCHSlots," + this.PUSCHSlots;
}
if (this.PUSCHPRB != null)
{
description += ",PUSCHPRB," + this.PUSCHPRB;
}
break;
case Format.WLAN_AC:
case Format.WLAN_AG:
case Format.WLAN_AX:
case Format.WLAN_B:
```

```
case Format.WLAN_BE:
case Format.WLAN_J:
case Format.WLAN_N:
case Format.WLAN_P:
if (this.ComboboxGuardIntervalSelectedValue != null)
{
description += ",GI," + ((double)this.ComboboxGuardIntervalSelectedValue).ToString(); //us
}
if (this.ComboboxMCSSelectedValue != null)
{
description += ",MCS," + ((int)this.ComboboxMCSSelectedValue).ToString();
}
if (this.ComboboxDataRateSelectedValue.Key != null &&
this.ComboboxDataRateSelectedValue.Key != DataRate.Undefined)
{
description += ",DataRate," + ((DataRate)this.ComboboxDataRateSelectedValue.Key).ToString();
}
if (this.ComboboxLTFCompTypeSelectedValue.Key != null &&
this.ComboboxLTFCompTypeSelectedValue.Key != LTFCompType.Undefined)
{
description += ",LTFType," +
((LTFCompType)this.ComboboxLTFCompTypeSelectedValue.Key).ToString();
}
if (this.DataSymbols != null)
{
description += ",DataSymbols," + ((int)this.DataSymbols).ToString();
}
break;
case Format.C2K:
if (this.ComboboxRadioConfigSelectedValue.Key != null &&
this.ComboboxRadioConfigSelectedValue.Key != RadioConfig.Undefined)
{
description += ",RConfig," +
((RadioConfig)this.ComboboxRadioConfigSelectedValue.Key).ToString();
}
break;
case Format._1xEvDo:
if (this.ComboboxPhysLayerSubtypeSelectedValue.Key != null &&
this.ComboboxPhysLayerSubtypeSelectedValue.Key != PhysicalLayerSubtype.Undefined)
{
description += ",PhysLayerSubtype," +
((PhysicalLayerSubtype)this.ComboboxPhysLayerSubtypeSelectedValue.Key).ToString();
```

```
}
break;
case Format.LTE_Downlink:
if (this.ComboboxLTEDownlinkModeSelectedValue.Key != null &&
this.ComboboxLTEDownlinkModeSelectedValue.Key != LTEDownlinkMode.Undefined)
{
description += ",DLMode," +
((LTEDownlinkMode)this.ComboboxLTEDownlinkModeSelectedValue.Key).ToString();
}
if (this.ULDLConfigIndex != null)
{
description += ",ULDLConfigIndex," + ((int)this.ULDLConfigIndex).ToString();
}
if (this.PDCCHNumSymbol != null)
{
description += ",PDCCHNumSymbol," + ((int)this.PDCCHNumSymbol).ToString();
}
break;
}
return description;
}

#endregion Save AIQ

#region Restore
public ICommand RestoreCommand
{
get
{
return new RelayCommand(restore, canRestore);
}
}

private void restore()
{
switch (this.TabSelectedIndex)
{
case 0:
restoreMarkerSegments(this.Marker1Segments, this.Marker1);
break;
case 1:
restoreMarkerSegments(this.Marker2Segments, this.Marker2);
```

```
break;
case 2:
restoreMarkerSegments(this.Marker3Segments, this.Marker3);
break;
case 3:
restoreMarkerSegments(this.Marker4Segments, this.Marker4);
break;
}
}

private void restoreMarkerSegments(ObservableCollection<MarkerSegment> segments, List<int>
marker)
{
segments.Clear();
MarkerSegment mkSeg = null;
for (int index = 0; index < marker.Count; index++)
{
determineMarkerSegment(marker[index], index, marker.Count - 1, segments, ref mkSeg);
}
}

private bool canRestore()
{
return true;
}
#endregion

#region Apply
public ICommand ApplyCommand
{
get
{
return new RelayCommand(apply, canApply);
}
}

private void apply()
{
switch (this.TabSelectedIndex)
{
case 0:
applyMarkerSegments(this.Marker1, this.Marker1Segments, 1);
```

```
OnChangeOccured();
break;
case 1:
applyMarkerSegments(this.Marker2, this.Marker2Segments, 2);
OnChangeOccured();
break;
case 2:
applyMarkerSegments(this.Marker3, this.Marker3Segments, 3);
OnChangeOccured();
break;
case 3:
applyMarkerSegments(this.Marker4, this.Marker4Segments, 4);
OnChangeOccured();
break;
}
}

private void applyMarkerSegments(List<int> marker, ObservableCollection<MarkerSegment>
segments, int chartSeriesIndex)
{
List<double> xVals = new List<double>(marker.Count);

for (int i = 0; i < marker.Count; i++)
{
marker[i] = 0;
xVals.Add(i);
}

foreach (MarkerSegment segment in segments)
{
for (int index = segment.From - 1; index <= segment.To - 1; index++)
{
marker[index] = 1;
}
}

redrawMarker(marker, chartSeriesIndex);
}

private bool canApply()
{
if (this.currentLoadedAiqFile == null && this.m_currentMTWaveform == null)
```

```
return false;

if (this.TabSelectedIndex == -1 || this.TabSelected == null)
return false;

string dgName = string.Format("dgMarker{0}", this.TabSelectedIndex + 1);

var dg = this.TabSelected.FindName(dgName) as DataGrid;

if (dg == null)
return false;

for (int i = 0; i < dg.Items.Count; i++)
{
DataGridRow row = dg.GetRow(i);
if (row == null)//unexpected
return false;
if (Validation.GetHasError(row))
return false;
}

return true;
}
#endregion

public void SaveOccuredSubscriber()
{
string ext = Path.GetExtension(this.InputFileName);

switch (ext)
{
case ".aiq":
SaveToAIQ(this.InputFileName);
break;
case ".mtwf2":
SaveToMTWF(this.InputFileName);
break;
}
}

#region Load MTWF
public string loadMtwf(string optFilename = null)
```

```
{
try
{
if (String.IsNullOrEmpty(optFilename))
{
// Configure open file dialog box
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();
dlg.DefaultExt = ".mtwf2"; // Default file extension
dlg.Filter = "Merlin Test Waveform File (.mtwf2)|*.mtwf2"; // Filter files by extension

// Show open file dialog box
var result = dlg.ShowDialog();
if (result == true)
{
this.InputFileName = dlg.FileName;
m_currentMTWaveform = MTWaveform.ImportMerlinTestWaveformBIN(dlg.FileName);
loadMtwfWorker();
View.SetupChart(null, "Samples", "Time (ms)", "Power (dBm)", 0, 0,
(double)this.NumberOfSamples, ((double)this.NumberOfSamples / (double)this.SamplingRate) /
1e3);
drawWholeChart();
return String.Format("Successfully loaded {0}", dlg.FileName);
}
else
{
return null;
}
}
else
{
this.InputFileName = optFilename;
m_currentMTWaveform = MTWaveform.ImportMerlinTestWaveformBIN(optFilename);
loadMtwfWorker();
View.SetupChart(null, "Samples", "Time (ms)", "Power (dBm)", 0, 0,
(double)this.NumberOfSamples, ((double)this.NumberOfSamples / (double)this.SamplingRate) /
1e3);
drawWholeChart();

return String.Format("Successfully loaded {0}", optFilename);
}
}
catch (Exception ex)
```

```
{
MessageBox.Show("An error occurred while loading .mtwf2 file: " + ex.Message, "Merlin Test
Studio", MessageBoxButton.OK, MessageBoxImage.Error);
return null;
}
}

private void loadMtwfWorker()
{
MetadataEditingEnabled = false;

// Clear previous files data.
Idata.Clear();
Qdata.Clear();
Power.Clear();
Marker1.Clear();
Marker2.Clear();
Marker3.Clear();
Marker4.Clear();
currentLoadedAiqFile = null;

clearMarkerSegments();

// Set Properties
// Common
this.ComboboxRadioFormatSelectedValue = ComboboxRadioFormatList.Where(a => a.Key ==
m_currentMTWaveform.Format).FirstOrDefault();
//Calculate CF on the fly for legacy mtwf formats where it's not stored, using RelRMS which is
stored and the fact that peak is 1.
if (double.IsNaN(m_currentMTWaveform.CF))
{
this.CrestFactor = (float)(20 * Math.Log10(1 / m_currentMTWaveform.RelRMS));
}
else
{
this.CrestFactor = (float)m_currentMTWaveform.CF;
}
// See below for duty cycle after IQ data comes in
this.ALCBandwidth = null;
// Setting the NumberOfSamples via the IData count is more compatible with older mtwf format
this.NumberOfSamples = m_currentMTWaveform.IData.Count;
this.RelRms = (float)m_currentMTWaveform.RelRMS;
```

```
this.Rms = null;
this.SamplingRate = m_currentMTWaveform.SR / 1e6;
this.SignalBandwidth = m_currentMTWaveform.BW / 1e6;
this.Description = null;
// Setting the TotalTime via the sampling rate and number of samples is more compatible with
older mtwf format
this.TotalTime = (m_currentMTWaveform.IData.Count / m_currentMTWaveform.SR) * 1e3;
this.Offset = m_currentMTWaveform.Offset * 1e3;
this.MeasLength = m_currentMTWaveform.MeasLen * 1e3;
this.StepLength = m_currentMTWaveform.StepLen * 1e3;
this.NumSteps = m_currentMTWaveform.NumSteps;

//NR
this.ComboboxSubcarrierSpacingSelectedValue = ComboboxSubcarrierSpacingList.Where(a => a
== m_currentMTWaveform.SCS / 1e3).FirstOrDefault(); //kHz
this.ComboboxCyclicPrefixSelectedValue = ComboboxCyclicPrefixList.Where(a => a.Key ==
m_currentMTWaveform.CP).FirstOrDefault();
this.ComboboxTransformPrecodingSelectedValue = ComboboxTransformPrecodingList.Where(a
=> a == m_currentMTWaveform.TPrecoding).FirstOrDefault();
this.ComboboxPUSCHModSelectedValue = ComboboxPUSCHModList.Where(a => a.Key ==
m_currentMTWaveform.PUSCHMod).FirstOrDefault();
this.PUSCHSlots = m_currentMTWaveform.PUSCHSlots;
this.PUSCHPRB = m_currentMTWaveform.PUSCHPRB;

//WLAN
this.ComboboxGuardIntervalSelectedValue = ComboboxGuardIntervalList.Where(a => a ==
m_currentMTWaveform.GI * 1e6).FirstOrDefault(); //us
this.ComboboxMCSSelectedValue = ComboboxMCSList.Where(a => a ==
m_currentMTWaveform.MCS).FirstOrDefault();
this.ComboboxDataRateSelectedValue = ComboboxDataRateList.Where(a => a.Key ==
m_currentMTWaveform.DataRate).FirstOrDefault();
this.ComboboxLTFCompTypeSelectedValue = ComboboxLTFCompTypeList.Where(a => a.Key
== m_currentMTWaveform.LTFType).FirstOrDefault();
this.DataSymbols = m_currentMTWaveform.DataSymbols;

//C2K
this.ComboboxRadioConfigSelectedValue = ComboboxRadioConfigList.Where(a => a.Key ==
m_currentMTWaveform.RConfig).FirstOrDefault();

//1xEVDO
this.ComboboxPhysLayerSubtypeSelectedValue = ComboboxPhysLayerSubtypeList.Where(a =>
a.Key == m_currentMTWaveform.PLSubtype).FirstOrDefault();
```

```
//LTE Downlink
this.ComboboxLTEDownlinkModeSelectedValue = ComboboxLTEDownlinkModeList.Where(a =>
a.Key == m_currentMTWaveform.DLMode).FirstOrDefault();
this.ULDLConfigIndex = m_currentMTWaveform.ConfigInd;
this.PDCCHNumSymbol = m_currentMTWaveform.PdcchNumSymbol;

for(int i = 0; i < m_currentMTWaveform.IData.Count; i++)
{
Idata.Add(Convert.ToDouble(m_currentMTWaveform.IData[i]));
Qdata.Add(Convert.ToDouble(m_currentMTWaveform.QData[i]));
double powerInWatts = m_currentMTWaveform.IData[i] * m_currentMTWaveform.IData[i] +
m_currentMTWaveform.QData[i] * m_currentMTWaveform.QData[i];

// Occasionaly we get IQ samples with a value of zero. To avoid an issue we just set the value
// to a really small value which avoid the issue when charting.
if (powerInWatts == 0D)
{
powerInWatts = 0.0000000000001;
}
// Store Result.
Power.Add(10.0 * Math.Log10(powerInWatts));
Marker1.Add(m_currentMTWaveform.Marker[i] & 0x1);
Marker2.Add(m_currentMTWaveform.Marker[i] & 0x2);
Marker3.Add(m_currentMTWaveform.Marker[i] & 0x4);
Marker4.Add(m_currentMTWaveform.Marker[i] & 0x8);
}

// Duty cycle can be updated if it wasn't available in metadata
if (double.IsNaN(m_currentMTWaveform.DC))
{
float finiteIQCount = 0;
for (int index = 0; index < this.Idata.Count; index++)
{
if (Math.Sqrt(this.Idata[index] * this.Idata[index] + this.Qdata[index] + this.Qdata[index]) != 0)
{
finiteIQCount += 1;
}
}
this.DutyCycle = (finiteIQCount / this.Idata.Count) * 100;
}
else
```

```
{
this.DutyCycle = (float)m_currentMTWaveform.DC;
}

MarkerSegment mk1Seg = null;
MarkerSegment mk2Seg = null;
MarkerSegment mk3Seg = null;
MarkerSegment mk4Seg = null;

for (int index = 0; index < Idata.Count; index++)
{
determineMarkerSegment(Marker1[index], index, Idata.Count - 1, this.Marker1Segments, ref
mk1Seg);
determineMarkerSegment(Marker2[index], index, Idata.Count - 1, this.Marker2Segments, ref
mk2Seg);
determineMarkerSegment(Marker3[index], index, Idata.Count - 1, this.Marker3Segments, ref
mk3Seg);
determineMarkerSegment(Marker4[index], index, Idata.Count - 1, this.Marker4Segments, ref
mk4Seg);
}

}

#endregion

#endregion Commands

#region Private Methods

private void drawWholeChart()
{
List<double> xActual = new List<double>(this.NumberOfSamples.Value);
for (int i = 0; i < this.NumberOfSamples.Value; i++)
xActual.Add(i);

const int POINTS_UPPER_LIMIT = 10000;

int total = xActual.Count;

if (total <= POINTS_UPPER_LIMIT)
{
double[] xValues = xActual.ToArray();
```

```
double[] xValuesTime = new double[xValues.Length];
double[] power = this.Power.ToArray();
double[] marker1 = this.Marker1.Select(m => m == 0 ? 0 : 1D).ToArray();
double[] marker2 = this.Marker2.Select(m => m == 0 ? 0 : 1D).ToArray();
double[] marker3 = this.Marker3.Select(m => m == 0 ? 0 : 1D).ToArray();
double[] marker4 = this.Marker4.Select(m => m == 0 ? 0 : 1D).ToArray();

for (int i = 0; i < xValues.Length; i++)
{
xValuesTime[i] = (xValues[i] / (double)this.SamplingRate) / 1e3;
}
this.View.DrawSeries(xValuesTime.ToArray(), power.ToArray(), 0, AxisType.Secondary);
this.View.DrawSeries(xValues, marker1, 1, AxisType.Primary);
this.View.DrawSeries(xValues, marker2, 2, AxisType.Primary);
this.View.DrawSeries(xValues, marker3, 3, AxisType.Primary);
this.View.DrawSeries(xValues, marker4, 4, AxisType.Primary);
}
else
{
List<double> xPValues = new List<double>(POINTS_UPPER_LIMIT * 2);
List<double> xPValuesTime = new List<double>(POINTS_UPPER_LIMIT * 2);
List<double> power = new List<double>(POINTS_UPPER_LIMIT * 2);

List<double> xMValues = new List<double>(POINTS_UPPER_LIMIT);
List<double> marker1 = new List<double>(POINTS_UPPER_LIMIT);
List<double> marker2 = new List<double>(POINTS_UPPER_LIMIT);
List<double> marker3 = new List<double>(POINTS_UPPER_LIMIT);
List<double> marker4 = new List<double>(POINTS_UPPER_LIMIT);

double batchSize = total / (double)POINTS_UPPER_LIMIT;

for (int i = 0; i < POINTS_UPPER_LIMIT; i++)
{
int batchFrom = (int)Math.Round(i * batchSize);
int batchTo = (int)Math.Round((i + 1) * batchSize) - 1;

//need???
if (batchTo > total - 1)
batchTo = total - 1;

getBound(this.Power, batchFrom, batchTo, out var maxPower, out var minPower, out int maxAt,
out int minAt);
```

```
if (maxAt > minAt)
{
xPValues.AddRange(new double[] { minAt, maxAt });
power.AddRange(new double[] { minPower, maxPower });
}
else
{
xPValues.AddRange(new double[] { maxAt, minAt });
power.AddRange(new double[] { maxPower, minPower });
}

xMValues.Add(i * batchSize);

marker1.Add(isAllZero(this.Marker1, batchFrom, batchTo) ? 0 : 1);
marker2.Add(isAllZero(this.Marker2, batchFrom, batchTo) ? 0 : 1);
marker3.Add(isAllZero(this.Marker3, batchFrom, batchTo) ? 0 : 1);
marker4.Add(isAllZero(this.Marker4, batchFrom, batchTo) ? 0 : 1);

}

//fix power start point
if (!xPValues.Contains(xActual[0]))
{
xPValues.Insert(0, xActual[0]);
power.Insert(0, this.Power[0]);
}
//fix power end point
if (!xPValues.Contains(xActual[total - 1]))
{
xPValues.Add(xActual[total - 1]);
power.Add(this.Power[total - 1]);
}

//fix marker end point
xMValues.Add(xActual[total - 1]);
marker1.Add(this.Marker1[total - 1] == 0 ? 0 : 1D);
marker2.Add(this.Marker2[total - 1] == 0 ? 0 : 1D);
marker3.Add(this.Marker3[total - 1] == 0 ? 0 : 1D);
marker4.Add(this.Marker4[total - 1] == 0 ? 0 : 1D);

for (int i = 0; i < xPValues.Count; i++)
{
```

```
xPValuesTime.Add((xPValues[i] / (double)this.SamplingRate) / 1e3);
}
this.View.DrawSeries(xPValuesTime.ToArray(), power.ToArray(), 0, AxisType.Secondary);
this.View.DrawSeries(xMValues.ToArray(), marker1.ToArray(), 1, AxisType.Primary);
this.View.DrawSeries(xMValues.ToArray(), marker2.ToArray(), 2, AxisType.Primary);
this.View.DrawSeries(xMValues.ToArray(), marker3.ToArray(), 3, AxisType.Primary);
this.View.DrawSeries(xMValues.ToArray(), marker4.ToArray(), 4, AxisType.Primary);
}
}

private void redrawMarker(List<int> listMarker, int chartSeriesIndex)
{
List<double> xActual = new List<double>(listMarker.Count);
for (int i = 0; i < listMarker.Count; i++)
xActual.Add(i);

const int POINTS_UPPER_LIMIT = 10000;

int total = xActual.Count;

if (total <= POINTS_UPPER_LIMIT)
{
double[] xValues = xActual.ToArray();
double[] marker = listMarker.Select(m => m == 0 ? 0 : 1D).ToArray();

this.View.DrawSeries(xValues, marker, chartSeriesIndex, AxisType.Primary);
}
else
{
List<double> xValues = new List<double>(POINTS_UPPER_LIMIT);
List<double> marker = new List<double>(POINTS_UPPER_LIMIT);

double batchSize = total / (double)POINTS_UPPER_LIMIT;

for (int i = 0; i < POINTS_UPPER_LIMIT; i++)
{
int batchFrom = (int)Math.Round(i * batchSize);
int batchTo = (int)Math.Round((i + 1) * batchSize) - 1;

//need???
if (batchTo > total - 1)
batchTo = total - 1;
```

```
xValues.Add(i * batchSize);

marker.Add(isAllZero(listMarker, batchFrom, batchTo) ? 0 : 1D);
}

//fix marker end point
xValues.Add(xActual[total - 1]);
marker.Add(this.Marker1[total - 1] == 0 ? 0 : 1D);

this.View.DrawSeries(xValues.ToArray(), marker.ToArray(), chartSeriesIndex, AxisType.Primary);
}
}

private void getBound(List<double> source, int indexFrom, int indexTo, out double max, out
double min, out int maxIndex, out int minIndex)
{
max = double.MinValue;
min = double.MaxValue;
maxIndex = -1;
minIndex = -1;
for (int i = indexFrom; i <= indexTo; i++)
{
var current = source[i];
if (current > max)
{
max = current;
maxIndex = i;
}
if (current < min)
{
min = current;
minIndex = i;
}
}
}

private bool isAllZero(List<int> source, int indexFrom, int indexTo)
{
for (int i = indexFrom; i <= indexTo; i++)
{
if (source[i] != 0)
```

```
return false;
}
return true;
}

#endregion Private Methods

//Example.
public event Action ChangeOccured; //Create action to pass out of MainWindowViewModel.

#region INotifyPropertyChanged

public event PropertyChangedEventHandler PropertyChanged;

/// <summary>
/// Create the OnPropertyChanged method to raise the event.
/// </summary>
/// <param name="name">Name of the property that's just been updated.</param>
private void OnPropertyChanged(string name)
{
PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null)
{
handler(this, new PropertyChangedEventArgs(name));
}
}

#endregion INotifyPropertyChanged

//Example to force ChangeOccured from anywhere in the project even from the user control.
public void OnChangeOccured()
{
ChangeOccured?.Invoke();
}
}

public class MarkerSegment : INotifyPropertyChanged, IEditableObject
{
private int from = 1;
private int to = 1;
```

```
/// <summary>
/// parameter-less contructor should be provided for datagrid to display the "new" row
/// </summary>
public MarkerSegment() { }

public int From
{
get => from;
set
{
from = value;
OnPropertyChanged("From");
}
}

public int To
{
get => to;
set
{
to = value;
OnPropertyChanged("To");
}
}

#region INotifyPropertyChanged
public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged(string name)
{
if (PropertyChanged != null)
{
PropertyChanged(this, new PropertyChangedEventArgs(name));
}
}
#endregion

#region IEditableObject
private MarkerSegment backupCopy;
private bool inEdit;

public void BeginEdit()
```

```
{
if (inEdit) return;
inEdit = true;
backupCopy = this.MemberwiseClone() as MarkerSegment;
}

public void CancelEdit()
{
if (!inEdit) return;
inEdit = false;
this.From = backupCopy.From;
this.To = backupCopy.To;
}

public void EndEdit()
{
if (!inEdit) return;
inEdit = false;
backupCopy = null;
}
#endregion

}

}
```

MarkerSegmentValidation.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;

namespace WaveformConverterControls
{
/// <summary>
/// ref:https://social.technet.microsoft.com/wiki/contents/articles/31422.wpf-passing-a-data-bound-value-to-a-validation-rule.aspx
```

```
/// </summary>
public class MarkerSegmentValidationRule : ValidationRule
{
public MarkerSegmentValidationData ValidationData { get; set; }
public override ValidationResult Validate(object value, CultureInfo cultureInfo)
{
var bg = value as BindingGroup;
if (bg != null)
{
foreach (var item in bg.Items)
{
var ms = item as MarkerSegment;
if (ms != null)
{
if (this.ValidationData != null)
{
if (ms.From < this.ValidationData.LowerLimit || ms.From > this.ValidationData.UpperLimit)
return new ValidationResult(false, string.Format("From should be between {0} and {1}",
this.ValidationData.LowerLimit, this.ValidationData.UpperLimit));
if (ms.To < this.ValidationData.LowerLimit || ms.To > this.ValidationData.UpperLimit)
return new ValidationResult(false, string.Format("To should be between {0} and {1}",
this.ValidationData.LowerLimit, this.ValidationData.UpperLimit));
}

if (ms.From > ms.To)
return new ValidationResult(false, "From should not be greater than To.");
}
}
}
return ValidationResult.ValidResult;
}
}

public class MarkerSegmentValidationData : DependencyObject
{
public static readonly DependencyProperty UpperLimitProperty =
DependencyProperty.Register("UpperLimit",
typeof(int),
typeof(MarkerSegmentValidationData));

public int UpperLimit
{
```

```
get => (int)GetValue(UpperLimitProperty);
set => SetValue(UpperLimitProperty, value);
}

public int LowerLimit { get; set; }
}

public class BindingProxy : Freezable
{
protected override Freezable CreateInstanceCore()
{
return new BindingProxy();
}

public object Data
{
get { return (object)GetValue(DataProperty); }
set { SetValue(DataProperty, value); }
}

public static readonly DependencyProperty DataProperty =
DependencyProperty.Register("Data",
typeof(object),
typeof(BindingProxy));
}

public static class DataGridExtension
{
public static DataGridRow GetRow(this DataGrid grid, int index)
{
DataGridRow row = grid.ItemContainerGenerator.ContainerFromIndex(index) as DataGridRow;
if (row == null)
{
// May be virtualized, bring into view and try again.
grid.UpdateLayout();
grid.ScrollIntoView(grid.Items[index]);
row = grid.ItemContainerGenerator.ContainerFromIndex(index) as DataGridRow;
}
return row;
}
}
}
```

Merlin Test Waveform.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace WaveformConverterControls
{
public enum Format
{
Undefined,
WLAN_AG,
WLAN_B,
WLAN_J,
WLAN_P,
WLAN_N,
WLAN_AC,
WLAN_AX,
WLAN_BE,
NR,
GSM_EDGE,
C2K,
_1xEvDo,
TDSCDMA,
WCDMA,
LTE_Uplink_FDD,
LTE_Uplink_TDD,
LTE_Downlink,
}

public enum DataRate
{
Undefined,
_1Mbps,
_2Mbps,
_5_5Mbps,
_11Mbps
}
```

```
public enum CyclicPrefix
{
Undefined,
Normal,
Extended
}

public enum PUSCHModulation
{
Undefined,
Pi_2_BPSK,
QPSK,
_16QAM,
_64QAM,
_256QAM
}

public enum RadioConfig
{
Undefined,
RC1,
RC2,
RC3,
RC4
}

public enum PhysicalLayerSubtype
{
Undefined,
Subtype0,
Subtype1,
Subtype2,
}

public enum AclrMode
{
Undefined,
UTRA,
EUTRA,
UTRA_EUTRA
}
```

```
public enum LTEDownlinkMode
{
Undefined,
FDD,
TDD
}

public enum LTFCompType
{
Undefined,
_4x,
_2x,
_1x
}

/// <summary>
/// A class to represent the merlin test waveform format.
/// </summary>
[Serializable]
public class MTWaveform
{
#region Constructor

/// <summary>
/// Constructor for the MTWaveform class.
/// </summary>
public MTWaveform()
{
// Set some defaults
#region General Properties

this.Format = Format.Undefined;
this.NSamples = double.NaN;
this.SR = double.NaN;
this.RelRMS = double.NaN;
this.BW = double.NaN;
this.Offset = double.NaN;
this.MeasLen = double.NaN;
this.StepLen = double.NaN;
this.NumSteps = -1;
this.DC = double.NaN;
```

```
this.CF = double.NaN;
this.WFLen = double.NaN;

#endregion General Properties

#region WLAN Metadata

this.GI = double.NaN;
this.MCS = -1;
this.DataRate = DataRate.Undefined;
this.LTFType = LTFCompType.Undefined;
this.DataSymbols = -1;

#endregion WLAN Metadata

#region NR Metadata

this.SCS = double.NaN;
this.CP = CyclicPrefix.Undefined;
this.TPrecoding = false;
this.PUSCHMod = PUSCHModulation.Undefined;
this.PUSCHSlots = string.Empty;
this.PUSCHPRB = string.Empty;

#endregion NR Metadata

#region C2K Metadata

this.RConfig = RadioConfig.Undefined;

#endregion C2K Metadata

#region 1xEVDO Metadata

this.PLSubtype = PhysicalLayerSubtype.Undefined;

#endregion 1xEVDO Metadata

#region LTEUplink Metadata

this.AclrMode = AclrMode.Undefined;
```

```
#endregion LTEUplink Metadata

#region LTEDownlink Metadata

this.DLMode = LTEDownlinkMode.Undefined;
this.ConfigInd = -1;
this.PdcchNumSymbol = -1;

#endregion LTEDownlink Metadata
}

#endregion Constructor

#region General Properties

/// <summary>
/// Header so that the binary file can be identified just from the binary data.
/// </summary>
public byte[] Header = new byte[] { 0xD, 0xE, 0xA, 0xD, 0xB, 0xE, 0xE, 0xF };

public int Ver { get; set; } = 1;

/// <summary>
/// Gets the radio format that this file represents.
/// </summary>
public Format Format { get; set; }

/// <summary>
/// Gets the number of IQ samples.
/// </summary>
public double NSamples { get; set; }

/// <summary>
/// Gets the sample rate at which the waveform will be played back at in Hz.
/// </summary>
public double SR { get; set; }

/// <summary>
/// Gets the RMS of the IQ.
/// </summary>
public double RelRMS { get; set; }
```

```
/// <summary>
/// Gets the bandwidth of the waveform in Hz.
/// </summary>
public double BW { get; set; }

/// <summary>
/// Gets the offset from the start of the measurement step to start making the power measurement in seconds. ( used to make an accurate power measurement on the waveform )
/// </summary>
public double Offset { get; set; }

/// <summary>
///  Gets the measurement length in seconds. ( used to make an accurate power measurement on the waveform )
/// </summary>
public double MeasLen { get; set; }

/// <summary>
/// Gets the step length in seconds. ( used to make an accurate power measurement on the waveform )
/// </summary>
public double StepLen { get; set; }

/// <summary>
/// Gets the number of steps. ( determines if the waveform is single/dual/quad site)
/// </summary>
public int NumSteps { get; set; }

/// <summary>
/// Gets the duty cycle of the waveform in %.
/// </summary>
public double DC { get; set; }

/// <summary>
/// Gets the crest factor of the waveform in dB.
/// </summary>
public double CF { get; set; }

/// <summary>
/// Gets the total length of the waveform in seconds.
/// </summary>
public double WFLen { get; set; }
```

```
/// <summary>
/// The I data portion of the waveform.
/// </summary>
public List<float> IData { get; set; } = new List<float>();

/// <summary>
/// The Q data portion of the waveform.
/// </summary>
public List<float> QData { get; set; } = new List<float>();

/// <summary>
/// The marker information per sample.
/// </summary>
public List<byte> Marker { get; set; } = new List<byte>();

#endregion General Properties

#region WLAN Metadata

/// <summary>
/// Gets the guard interval of the waveform in seconds.
/// </summary>
public double GI { get; set; }

/// <summary>
/// Gets the MCS index of the waveform.
/// </summary>
public int MCS { get; set; }

/// <summary>
/// Gets the data rate of the waveform.
/// </summary>
public DataRate DataRate { get; set; }

/// <summary>
/// Gets the LTF compression type of the waveform.
/// </summary>
public LTFCompType LTFType { get; set; }

/// <summary>
/// Gets the number of Data symbols of the waveform.
```

```
/// </summary>
public int DataSymbols { get; set; }

#endregion WLAN Metadata

#region NR Metadata
/// <summary>
/// Gets the subcarrier spacing in Hz.
/// </summary>
public double SCS { get; set; }

/// <summary>
/// Gets the cyclic prefix (Normal/Extended).
/// </summary>
public CyclicPrefix CP { get; set; }

/// <summary>
/// Gets whether waveform genereated with transform precoding (DFT-s-OFDM) or not (CP-OFDM).
/// </summary>
public bool TPrecoding { get; set; }

/// <summary>
/// Gets the modulation of the waveform.
/// </summary>
public PUSCHModulation PUSCHMod { get; set; }

/// <summary>
/// Gets the allocated slots (time domain) of the waveform.
/// </summary>
public string PUSCHSlots { get; set; } = string.Empty;

/// <summary>
/// Gets the allocated resource blocks (frequency domain) of the waveform.
/// </summary>
public string PUSCHPRB { get; set; } = string.Empty;



#endregion NR Metadata

#region C2K Metadata
/// <summary>
```

```
/// Gets the radio configuration.
/// </summary>
public RadioConfig RConfig { get; set; }

#endregion C2K Metadata

#region 1xEVDO Metadata
/// <summary>
/// Gets the physical layer subtype.
/// </summary>
public PhysicalLayerSubtype PLSubtype { get; set; }

#endregion 1xEVDO Metadata

#region LTEUplink Metadata
/// <summary>
/// Gets the ACLR Mode.
/// </summary>
public AclrMode AclrMode { get; set; } // could make the argument this doesn't belong here since it's really a measurement setting
#endregion LTEUplink Metadata

#region LTEDownlink Metadata
/// <summary>
/// Gets the downlink mode (FDD/TDD).
/// </summary>
public LTEDownlinkMode DLMode { get; set; }

/// <summary>
/// Gets the uplink/downlink configuration index.
/// </summary>
public int ConfigInd { get; set; }

/// <summary>
/// Gets the physical downlink control channel number symbol.
/// </summary>
public int PdcchNumSymbol { get; set; }
#endregion LTEDownlink Metadata

#region Import / Save Methods

/// <summary>
```

```
/// Writes the current object to binary, new format.
/// </summary>
/// <param name="Filename">The filename and path of the file to write.</param>
public void SaveToBinary(string Filename)
{
//Write out in Merlin Test Binary Format.
using (BinaryWriter bw = new BinaryWriter(File.Open(Filename, FileMode.Create)))
{
#region General Properties

bw.Write(this.Header);
bw.Write(this.Ver);
bw.Write((int)this.Format);
bw.Write(this.NSamples);
bw.Write(this.SR / 1e6);        //MHz
bw.Write(this.RelRMS);
bw.Write(this.BW / 1e6);        //MHz
bw.Write(this.Offset * 1e3);    //ms
bw.Write(this.MeasLen * 1e3);   //ms
bw.Write(this.StepLen * 1e3);   //ms
bw.Write(this.NumSteps);
bw.Write(this.DC);
bw.Write(this.CF);
bw.Write(this.WFLen * 1e3);     //ms

#endregion General Properties

#region WLAN Metadata

bw.Write(GI * 1e6);             //us
bw.Write(MCS);
bw.Write((int)DataRate);
bw.Write((int)LTFType);
bw.Write(DataSymbols);

#endregion WLAN Metadata

#region NR Metadata

bw.Write(SCS / 1e3);            //kHz
bw.Write((int)CP);
bw.Write(TPrecoding);
```

```
bw.Write((int)PUSCHMod);
bw.Write(PUSCHSlots);
bw.Write(PUSCHPRB);

#endregion NR Metadata

#region C2K Metadata

bw.Write((int)RConfig);

#endregion C2K Metadata

#region 1xEVDO Metadata

bw.Write((int)PLSubtype);

#endregion 1xEVDO Metadata

#region LTEUplink Metadata

bw.Write((int)AclrMode);

#endregion LTEUplink Metadata

#region LTEDownlink Metadata

bw.Write((int)DLMode);
bw.Write(ConfigInd);
bw.Write(PdcchNumSymbol);

#endregion LTEDownlink Metadata

#region IQ and Marker

// Write out I,Q and Marker data.
for (int index = 0; index < this.IData.Count; index++)
{
bw.Write(this.IData[index]);
bw.Write(this.QData[index]);
bw.Write(this.Marker[index]);
}
```

```
#endregion IQ and Marker

bw.Flush();
}

}

/// <summary>
/// Helper method that determines if a file is the old or the new file format.
/// </summary>
/// <param name="FileName">Filename of the file to be tested.</param>
/// <returns>True if it's the new file format and false if it is not.</returns>
private static bool IsNewFormat(string FileName)
{
int[] pattern = new int[] { 0xD, 0xE, 0xA, 0xD, 0xB, 0xE, 0xE, 0xF };
int searchPosition = 0;

using (FileStream fs = new FileStream(FileName, FileMode.Open))
{
// Keep searching until we get to the end of file or we've gone through the first 2000 bytes.
while (fs.Position < fs.Length && fs.Position < 8)
{

var latestbyte = fs.ReadByte();

if (latestbyte == pattern[searchPosition])
{
searchPosition++;

if (searchPosition == pattern.Length)
{
return true;
}
}
else
{
searchPosition = 0;
}

if (latestbyte == -1) return false; //We have reached the end of the file
}
```

```
return false;
}
}

/// <summary>
/// Imports a Merlin Test Waveform in legacy or new format.
/// </summary>
/// <param name="FileName">The filename and path of the file.</param>
/// <returns>A Merlin Test Waveform object.</returns>
public static MTWaveform ImportMerlinTestWaveformBIN(string FileName)
{
MTWaveform waveform = null;

// Determine if this is the older fileformat ( without meta data )
if (IsNewFormat(FileName) == true)
{
using (FileStream fs = new FileStream(FileName, FileMode.Open, FileAccess.Read))
{
using (BinaryReader br = new BinaryReader(fs))
{
waveform = new MTWaveform();

fs.Seek(8, SeekOrigin.Current);

waveform.Ver = br.ReadInt32();

switch (waveform.Ver)
{
case 1:
{
#region General Properties

waveform.Format = (Format)br.ReadInt32();
waveform.NSamples = br.ReadDouble();
waveform.SR = br.ReadDouble() * 1e6;
waveform.RelRMS = br.ReadDouble();
waveform.BW = br.ReadDouble() * 1e6;
waveform.Offset = br.ReadDouble() / 1e3;
waveform.MeasLen = br.ReadDouble() / 1e3;
waveform.StepLen = br.ReadDouble() / 1e3;
waveform.NumSteps = br.ReadInt32();
waveform.DC = br.ReadDouble();
```

```
waveform.CF = br.ReadDouble();
waveform.WFLen = br.ReadDouble() / 1e3;

#endregion General Properties

#region WLAN Metadata

waveform.GI = br.ReadDouble() / 1e6;
waveform.MCS = br.ReadInt32();
waveform.DataRate = (DataRate)br.ReadInt32();
waveform.LTFType = (LTFCompType)br.ReadInt32();
waveform.DataSymbols = br.ReadInt32();

#endregion WLAN Metadata

#region NR Metadata

waveform.SCS = br.ReadDouble() * 1e3;
waveform.CP = (CyclicPrefix)br.ReadInt32();
waveform.TPrecoding = br.ReadBoolean();
waveform.PUSCHMod = (PUSCHModulation)br.ReadInt32();
waveform.PUSCHSlots = br.ReadString();
waveform.PUSCHPRB = br.ReadString();

#endregion NR Metadata

#region C2K Metadata

waveform.RConfig = (RadioConfig)br.ReadInt32();

#endregion C2K Metadata

#region 1xEVDO Metadata

waveform.PLSubtype = (PhysicalLayerSubtype)br.ReadInt32();

#endregion 1xEVDO Metadata

#region LTEUplink Metadata

waveform.AclrMode = (AclrMode)br.ReadInt32();
```

```
#endregion LTEUplink Metadata

#region LTEDownlink Metadata

waveform.DLMode = (LTEDownlinkMode)br.ReadInt32();
waveform.ConfigInd = br.ReadInt32();
waveform.PdcchNumSymbol = br.ReadInt32();

#endregion LTEDownlink Metadata

#region IQ and marker Data

while (br.BaseStream.Position < br.BaseStream.Length)
{
var iValue = br.ReadSingle();
var qValue = br.ReadSingle();
var markerByte = br.ReadByte();

waveform.IData.Add(iValue);
waveform.QData.Add(qValue);
waveform.Marker.Add(markerByte);
}

#endregion IQ and marker Data
break;
}
default:
{
throw new NotSupportedException("Binary format version not supported.");
}
}

}
}
}
else
{
using (FileStream fs = new FileStream(FileName, FileMode.Open, FileAccess.Read))
{
using (BinaryReader br = new BinaryReader(fs))
{
waveform = new MTWaveform();
```

```
waveform.SR = br.ReadDouble(); //Sample rate already stored in Hz for legacy format, no need to convert.
waveform.RelRMS = br.ReadDouble();

while (br.BaseStream.Position < br.BaseStream.Length)
{
var iValue = br.ReadDouble();
var qValue = br.ReadDouble();
var markerByte = br.ReadByte();

waveform.IData.Add((float)iValue);
waveform.QData.Add((float)qValue);
waveform.Marker.Add(markerByte);
}
}
}
}

return waveform;
}

/// <summary>
/// Imports a Merlin Test Waveform in .txt format.
/// </summary>
/// <param name="FileName">The filename and path of the file.</param>
/// <returns>A Merlin Test Waveform object.</returns>
public static MTWaveform ImportMerlinTestWaveformMatlab(string FileName)
{
MTWaveform waveformObj = new MTWaveform();
string line;
bool headerComplete = false;

using (StreamReader sr = new StreamReader(FileName))
{
while ((line = sr.ReadLine()) != null)
{
var split = line.Split(',');

if (split.Length != 2)
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
```

```
if (headerComplete == false)
{
switch (split[0])
{
case "Ver":
{
if (int.TryParse(split[1], out int value) == true)
{
waveformObj.Ver = value;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "Format":
{
if (Enum.TryParse(split[1], out Format readInformat) == true)
{
waveformObj.Format = readInformat;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "NSamples":
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.NSamples = value;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "SR":
```

```
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.SR = value * 1e6;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "RelRMS":
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.RelRMS = value;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "BW":
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.BW = value * 1e6;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "Offset":
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.Offset = value / 1e3;
}
else
```

```
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "MeasLen":
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.MeasLen = value / 1e3;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "StepLen":
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.StepLen = value / 1e3;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "NumSteps":
{
if (int.TryParse(split[1], out int value) == true)
{
waveformObj.NumSteps = value;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "DC":
```

```
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.DC = value;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "CF":
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.CF = value;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "WFLen":
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.WFLen = value / 1e3;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "MCS":
{
if (int.TryParse(split[1], out int value) == true)
{
waveformObj.MCS = value;
}
else
```

```
{
if (split[1].Contains("N/A") == true)
{
waveformObj.MCS = -1;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
}
break;
}
case "DataRate":
{
if (Enum.TryParse(split[1], out DataRate value) == true)
{
waveformObj.DataRate = value;
}
else
{
if (split[1].Contains("N/A") == true)
{
waveformObj.DataRate = DataRate.Undefined;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
}
break;
}
case "GI":
{
if (double.TryParse(split[1], out double value) == true)
{
waveformObj.GI = value / 1e6;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
```

```
}
case "LTFType":
{
if (Enum.TryParse(split[1], out LTFCompType value) == true)
{
waveformObj.LTFType = value;
}
else
{
if (split[1].Contains("N/A") == true)
{
waveformObj.LTFType = LTFCompType.Undefined;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
}
break;
}
case "DataSymbols":
{
if (int.TryParse(split[1], out int value) == true)
{
waveformObj.DataSymbols = value;
}
else
{
if (split[1].Contains("N/A") == true)
{
waveformObj.DataSymbols = -1;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
}
break;
}
case "SCS":
{
if (double.TryParse(split[1], out double value) == true)
```

```
{
waveformObj.SCS = value * 1e3;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "CP":
{
if (Enum.TryParse(split[1], out CyclicPrefix value) == true)
{
waveformObj.CP = value;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "TPrecoding":
{
if (bool.TryParse(split[1], out bool value) == true)
{
waveformObj.TPrecoding = value;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
break;
}
case "PUSCHMod":
{
if (Enum.TryParse(split[1], out PUSCHModulation value) == true)
{
waveformObj.PUSCHMod = value;
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
```

```
}
break;
}
case "PUSCHSlots":
{
waveformObj.PUSCHSlots = split[1];
break;
}
case "PUSCHPRB":
{
waveformObj.PUSCHPRB = split[1];
break;
}
case "IQ":
{
// Once we reach an "IQ" we know that the following lines are just interleaved IQ amd so we can mark the header as complete.
headerComplete = true;
break;
}
default:
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}
}
}
else
{
// I Data
if (float.TryParse(split[0], out float ivalue) == true)
{
waveformObj.IData.Add(ivalue);
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}

//Q Data
if (float.TryParse(split[1], out float qvalue) == true)
{
waveformObj.QData.Add(qvalue);
```

```
}
else
{
throw new FormatException("File is not a merlin test matlab meta data file!");
}

}
}
}

return waveformObj;
}



#endregion Import / Save Methods
}
}
```

RelayCommands.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
using System.Windows.Input;

namespace WaveformConverterControls
{
//  Original author - Josh Smith - http://msdn.microsoft.com/en-
us/magazine/dd419663.aspx#id0090030

/// <summary>
/// A command whose sole purpose is to relay its functionality to other objects by invoking
delegates. The default return value for the CanExecute method is 'true'.
/// </summary>
public class RelayCommand<T> : ICommand
{

#region Declarations

readonly Predicate<T> _canExecute;
readonly Action<T> _execute;
```

```
#endregion

#region Constructors

/// <summary>
/// Initializes a new instance of the <see cref="RelayCommand&lt;T&gt;"/> class and the command can always be executed.
/// </summary>
/// <param name="execute">The execution logic.</param>
public RelayCommand(Action<T> execute)
: this(execute, null)
{
}

/// <summary>
/// Initializes a new instance of the <see cref="RelayCommand&lt;T&gt;"/> class.
/// </summary>
/// <param name="execute">The execution logic.</param>
/// <param name="canExecute">The execution status logic.</param>
public RelayCommand(Action<T> execute, Predicate<T> canExecute)
{

if (execute == null)
throw new ArgumentNullException("execute");
_execute = execute;
_canExecute = canExecute;
}

#endregion

#region ICommand Members

public event EventHandler CanExecuteChanged
{
add
{

if (_canExecute != null)
CommandManager.RequerySuggested += value;
}
remove
```

```
{

if (_canExecute != null)
CommandManager.RequerySuggested -= value;
}
}

[DebuggerStepThrough]
public Boolean CanExecute(Object parameter)
{
return _canExecute == null ? true : _canExecute((T)parameter);
}

public void Execute(Object parameter)
{
_execute((T)parameter);
}

#endregion
}

/// <summary>
/// A command whose sole purpose is to relay its functionality to other objects by invoking delegates. The default return value for the CanExecute method is 'true'.
/// </summary>
public class RelayCommand : ICommand
{

#region Declarations

readonly Func<Boolean> _canExecute;
readonly Action _execute;

#endregion

#region Constructors

/// <summary>
/// Initializes a new instance of the <see cref="RelayCommand&lt;T&gt;"/> class and the command can always be executed.
/// </summary>
/// <param name="execute">The execution logic.</param>
```

```
public RelayCommand(Action execute)
: this(execute, null)
{
}

/// <summary>
/// Initializes a new instance of the <see cref="RelayCommand&lt;T&gt;"/> class.
/// </summary>
/// <param name="execute">The execution logic.</param>
/// <param name="canExecute">The execution status logic.</param>
public RelayCommand(Action execute, Func<Boolean> canExecute)
{

if (execute == null)
throw new ArgumentNullException("execute");
_execute = execute;
_canExecute = canExecute;
}

#endregion

#region ICommand Members

public event EventHandler CanExecuteChanged
{
add
{

if (_canExecute != null)
CommandManager.RequerySuggested += value;
}
remove
{

if (_canExecute != null)
CommandManager.RequerySuggested -= value;
}
}

[DebuggerStepThrough]
public Boolean CanExecute(Object parameter)
{
```

```
return _canExecute == null ? true : _canExecute();
}

public void Execute(Object parameter)
{
_execute();
}

#endregion
}
}
```

StopWatchNs.cs

```
using System;
using System.Collections.Generic;

using System.Text;
using System.Diagnostics;

namespace WaveformConverterControls
{
/// <summary>
/// A high precision stopwatch.
/// </summary>
public class StopWatchNs : Stopwatch
{
/// <summary>
/// Represents the nanno seconds per clock tick.
/// </summary>
private double nanosecPerTick;

/// <summary>
/// StopwatchNs constructor.
/// </summary>
public StopWatchNs()
{
long frequency = Stopwatch.Frequency;

nanosecPerTick = (double)(1000L * 1000L * 1000L) / (double)frequency;

}
```

```
/// <summary>
/// Creates a new instance of the StopWatchNs class and starts the timer.
/// </summary>
/// <returns></returns>
public new static StopWatchNs StartNew()
{
StopWatchNs tmp = new StopWatchNs();
tmp.Start();

return tmp;
}

/// <summary>
/// Gets the elapsed time in milleseconds.
/// </summary>
public new double ElapsedMilliseconds
{
get
{
return ((double)ElapsedNanoSeconds / 1000000);
}
}

/// <summary>
/// Gets the elapsed time in nanno seconds.
/// </summary>
public double ElapsedNanoSeconds
{
get
{
return nanosecPerTick * this.ElapsedTicks;
}
}

}
}
```

WaveformConverterControl.xaml.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Forms.DataVisualization.Charting;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WaveformConverterControls
{
/// <summary>
/// Interaction logic for WaveformConverterControl.xaml
/// </summary>
public partial class WaveformConverterControl : UserControl, IView
{
#region Private Member Variables

/// <summary>
/// Instance of this VIEWs VIEW-MODEL;
/// </summary>
private MainWindowViewModel viewModelObj;

#endregion Private Member Variables

//public string FilePathToLoad
//{
//    get { return (string)GetValue(Property1Property); }
//    set { SetValue(Property1Property, value); }
//}
//
//// Using a DependencyProperty as the backing store for Property1.
//// This enables animation, styling, binding, etc...
//public static readonly DependencyProperty Property1Property
//    = DependencyProperty.Register(
//          "Property1",
```

```
//          typeof(string),
//          typeof(WaveformConverterControl),
//          new PropertyMetadata(false)
//      );

public WaveformConverterControl(MainWindowViewModel dataContext)
{
InitializeComponent();

this.DataContext = dataContext;

//  We have declared the view model instance declaratively in the xaml.
//  Get the reference to it here.
viewModelObj = dataContext;

// Subscribe to the VIEW's loaded event and execute a method that in turn will execute a method in the View-Model when the View is loaded.
//this.Loaded += new RoutedEventHandler(MainWindow_Loaded);

// Set the ViewModels view property to be equal to this. This allows the viewModel access to the view but only
// through a controled interface thus not breaking the MVVM pattern.
viewModelObj.View = this as IView;
}

public WaveformConverterControl()
{
InitializeComponent();

this.DataContext = new MainWindowViewModel();

//  We have declared the view model instance declaratively in the xaml.
//  Get the reference to it here.
viewModelObj = (DataContext as MainWindowViewModel);

// Subscribe to the VIEW's loaded event and execute a method that in turn will execute a method in the View-Model when the View is loaded.
//this.Loaded += new RoutedEventHandler(MainWindow_Loaded);

// Set the ViewModels view property to be equal to this. This allows the viewModel access to the view but only
// through a controled interface thus not breaking the MVVM pattern.
```

```
viewModelObj.View = this as IView;
}

public void SetupChart(string Title, string XaxisTitle, string Xaxis2Title, string YaxisTitle, double XaxisMinimum, double Xaxis2Minimum, double XaxisMaximum, double Xaxis2Maximum)
{
panel1Chart.SetupChart(Title, XaxisTitle, Xaxis2Title, YaxisTitle, XaxisMinimum, Xaxis2Minimum, XaxisMaximum, Xaxis2Maximum);
}

public void DrawSeries(double[] xValues, double[] results, int seriesIndex, AxisType axisType)
{
panel1Chart.DrawSeries(xValues, results, seriesIndex, axisType);
}

private void UserControl_Loaded(object sender, RoutedEventArgs e)
{

}
}

}
```

XvsYGenericChart.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Reflection;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Forms.DataVisualization.Charting;
using System.Windows.Media;
using System.Windows.Media.Imaging;

namespace WaveformConverterControls
{
public class Stat
{
public double Max1 { get; set; }
public double Max2 { get; set; }
public double Min1 { get; set; }
```

```
public double Min2 { get; set; }
public double Average1 { get; set; }
public double Average2 { get; set; }
public double StdDev1 { get; set; }
public double StdDev2 { get; set; }
public string Units { get; set; }
}
/// <summary>
/// Class that contains extension methods to the MSChart Control.
/// </summary>
internal static class MSChartExtension
{
/// <summary>
/// Extension method that clears all the points in a series. ( the official method is very slow. )
/// </summary>
/// <param name="sender">The series to clear points </param>
public static void ClearPoints(this System.Windows.Forms.DataVisualization.Charting.Series
sender)
{
// https://www.codeproject.com/Articles/376060/Speedup-MSChart-Clear-Data-Points
sender.Points.SuspendUpdates();
while (sender.Points.Count > 0)
sender.Points.RemoveAt(sender.Points.Count - 1);
sender.Points.ResumeUpdates();
sender.Points.Clear(); //NOTE 1
}
}
/// <summary>
/// Interaction logic for PowerVsTime.xaml
/// </summary>
public partial class XvsYGenericChart : UserControl
{
#region Private Member Variables
private const string DEFAULT_CHART_AREA = "DefaultChartArea";
private const string Marker_CHART_AREA = "MarkerChartArea";
private const string DEFAULT_LEGEND = "DefaultLegend";
private const string DEFAULT_TITLE = "DefaultTitle";
private const string IMAGE_NAME_CHECKED = "ifchecked";
private const string IMAGE_NAME_UNCHECKED = "ifunchecked";
#endregion Private Member Variables

#region Constructor
```

```
/// <summary>
/// Constructor for the XvsYGeneric class.
/// </summary>
public XvsYGenericChart()
{
InitializeComponent();

Chart chartObj = this.FindName("XvsYChart") as Chart;

Assembly assembly = Assembly.GetExecutingAssembly();
chartObj.Images.Add(new NamedImage(IMAGE_NAME_CHECKED,
System.Drawing.Image.FromStream(assembly.GetManifestResourceStream("WaveformConverter
Controls.if_checked_checkbox.png"))));
chartObj.Images.Add(new NamedImage(IMAGE_NAME_UNCHECKED,
System.Drawing.Image.FromStream(assembly.GetManifestResourceStream("WaveformConverter
Controls.if_unchecked_checkbox.png"))));

//default chart area
var defaultArea = chartObj.ChartAreas.Add(DEFAULT_CHART_AREA);
{
defaultArea.CursorX.Interval = 0;
defaultArea.CursorX.IsUserEnabled = true;
defaultArea.CursorX.IsUserSelectionEnabled = true;
defaultArea.CursorX.LineColor = System.Drawing.Color.White;
defaultArea.CursorX.SelectionColor = System.Drawing.Color.Gray;

defaultArea.AxisX2.TitleFont = new Font("Tahoma", 10, System.Drawing.FontStyle.Bold);
defaultArea.AxisX2.TitleForeColor = System.Drawing.Color.WhiteSmoke;
defaultArea.AxisX2.LabelStyle.ForeColor = System.Drawing.Color.Gray;
defaultArea.AxisX2.LabelStyle.Format = "{0.###}";
defaultArea.AxisX2.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisX2.MinorGrid.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisX2.MajorGrid.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisX2.MajorTickMark.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisX2.IsStartedFromZero = true;
defaultArea.AxisX2.IsMarginVisible = false;

defaultArea.AxisY.TitleFont = new Font("Tahoma", 10, System.Drawing.FontStyle.Bold);
defaultArea.AxisY.TitleForeColor = System.Drawing.Color.WhiteSmoke;
defaultArea.AxisY.LabelStyle.ForeColor = System.Drawing.Color.Gray;
defaultArea.AxisY.LineColor = System.Drawing.Color.Gray;
```

```
defaultArea.AxisY.MinorGrid.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisY.MajorGrid.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisY.MajorTickMark.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisY.IsStartedFromZero = false;

defaultArea.BackColor = chartObj.BackColor;
defaultArea.Visible = false;
}

var markerArea = chartObj.ChartAreas.Add(Marker_CHART_AREA);
{
markerArea.CursorX.Interval = 0;
markerArea.CursorX.IsUserEnabled = true;
markerArea.CursorX.IsUserSelectionEnabled = true;
markerArea.CursorX.LineColor = System.Drawing.Color.White;
markerArea.CursorX.SelectionColor = System.Drawing.Color.Gray;

markerArea.AxisX.TitleFont = new Font("Tahoma", 10, System.Drawing.FontStyle.Bold);
markerArea.AxisX.TitleForeColor = System.Drawing.Color.WhiteSmoke;
markerArea.AxisX.LabelStyle.ForeColor = System.Drawing.Color.Gray;
markerArea.AxisX.LineColor = System.Drawing.Color.Gray;
markerArea.AxisX.MinorGrid.LineColor = System.Drawing.Color.Gray;
markerArea.AxisX.MajorGrid.LineColor = System.Drawing.Color.Gray;
markerArea.AxisX.MajorTickMark.LineColor = System.Drawing.Color.Gray;
markerArea.AxisX.IsStartedFromZero = true;
markerArea.AxisX.IsMarginVisible = false;

markerArea.AxisY.TitleFont = new Font("Tahoma", 10, System.Drawing.FontStyle.Bold);
markerArea.AxisY.TitleForeColor = System.Drawing.Color.WhiteSmoke;
markerArea.AxisY.LabelStyle.ForeColor = System.Drawing.Color.Gray;
markerArea.AxisY.LineColor = System.Drawing.Color.Gray;
markerArea.AxisY.MinorGrid.LineColor = System.Drawing.Color.Gray;
markerArea.AxisY.MajorGrid.LineColor = System.Drawing.Color.Gray;
markerArea.AxisY.MajorTickMark.LineColor = System.Drawing.Color.Gray;
markerArea.AxisY.IsStartedFromZero = false;

//This will align inner plot area of marker graph to power graph
markerArea.AlignWithChartArea = defaultArea.Name;
markerArea.AlignmentStyle = AreaAlignmentStyles.PlotPosition;
markerArea.AlignmentOrientation = AreaAlignmentOrientations.Vertical;

markerArea.BackColor = chartObj.BackColor;
```

```
markerArea.Visible = false;
}

// default legend
var defaultLegend = chartObj.Legends.Add(DEFAULT_LEGEND);
{
defaultLegend.BackColor = System.Drawing.Color.Gray;
defaultLegend.BorderColor = System.Drawing.Color.Black;
defaultLegend.BorderWidth = 2;
defaultLegend.BorderDashStyle = ChartDashStyle.Solid;
defaultLegend.ShadowOffset = 2;
defaultLegend.Alignment = StringAlignment.Center;
defaultLegend.LegendStyle = LegendStyle.Table;
defaultLegend.Docking = Docking.Bottom;
//defaultLegend.DockedToChartArea = DEFAULT_CHART_AREA;
defaultLegend.IsDockedInsideChartArea = false;
}

// title
var title = chartObj.Titles.Add(string.Empty);
{
title.Name = DEFAULT_TITLE;
title.Font = new Font("Tahoma", 14, System.Drawing.FontStyle.Bold);
title.ForeColor = System.Drawing.Color.WhiteSmoke;
title.Docking = Docking.Top;
title.DockedToChartArea = DEFAULT_CHART_AREA;
title.IsDockedInsideChartArea = false;
}

// series
chartObj.Series.Add("Power").Color = System.Drawing.Color.Red;
chartObj.Series.Add("Marker 1").Color = System.Drawing.Color.Green;
chartObj.Series.Add("Marker 2").Color = System.Drawing.Color.Blue;
chartObj.Series.Add("Marker 3").Color = System.Drawing.Color.Yellow;
chartObj.Series.Add("Marker 4").Color = System.Drawing.Color.Cyan;

foreach (var series in chartObj.Series)
{
series.IsVisibleInLegend = false;
series.IsValueShownAsLabel = false;
if (series.Name == "Power")
{
```

```
series.ChartArea = DEFAULT_CHART_AREA;
}
else
{
series.ChartArea = Marker_CHART_AREA;
}
series.ChartType = SeriesChartType.FastLine;
series.SmartLabelStyle.Enabled = false;
series.Enabled = false;

LegendItem legendItem = new LegendItem
{
ImageStyle = LegendImageStyle.Rectangle,
Color = series.Color,
BorderColor = series.Color
};
legendItem.Cells.Add(LegendCellType.Image, IMAGE_NAME_UNCHECKED,
ContentAlignment.MiddleCenter);
legendItem.Cells.Add(LegendCellType.SeriesSymbol, series.Name,
ContentAlignment.MiddleCenter);
legendItem.Cells.Add(LegendCellType.Text, series.Name, ContentAlignment.MiddleLeft);

legendItem.Tag = series;

series.Tag = legendItem;

defaultLegend.CustomItems.Add(legendItem);
}

}

#endregion Constructor

#region Public Properties

/// <summary>
/// Gets / Sets the title of the chart.
/// </summary>
public string Title
{
get
{
```

```
Chart chart = this.FindName("XvsYChart") as Chart;
return chart.Titles[DEFAULT_TITLE].Text;
}
set
{
Chart chart = this.FindName("XvsYChart") as Chart;
chart.Titles[DEFAULT_TITLE].Text = value;
}
}

/// <summary>
/// Gets / Sets the X axis title of the chart.
/// </summary>
public string TitleXAxis
{
get
{
Chart chart = this.FindName("XvsYChart") as Chart;
return chart.ChartAreas[DEFAULT_CHART_AREA].AxisX.Title;
}
set
{
Chart chart = this.FindName("XvsYChart") as Chart;
chart.ChartAreas[DEFAULT_CHART_AREA].AxisX.Title = value;
chart.ChartAreas[Marker_CHART_AREA].AxisX.Title = value;
}
}

/// <summary>
/// Gets / Sets the secondary X axis title of the chart.
/// </summary>
public string TitleXAxis2
{
get
{
Chart chart = this.FindName("XvsYChart") as Chart;
return chart.ChartAreas[DEFAULT_CHART_AREA].AxisX2.Title;
}
set
{
Chart chart = this.FindName("XvsYChart") as Chart;
chart.ChartAreas[DEFAULT_CHART_AREA].AxisX2.Title = value;
```

```
chart.ChartAreas[Marker_CHART_AREA].AxisX2.Title = value;
}
}

/// <summary>
/// Gets / Sets the Y axis title of the chart.
/// </summary>
public string TitleYAxis
{
get
{
Chart chart = this.FindName("XvsYChart") as Chart;
return chart.ChartAreas[DEFAULT_CHART_AREA].AxisY.Title;
}
set
{
Chart chart = this.FindName("XvsYChart") as Chart;
chart.ChartAreas[DEFAULT_CHART_AREA].AxisY.Title = value;
chart.ChartAreas[Marker_CHART_AREA].AxisY.Title = "Marker On/Off";
}
}

#endregion Public Properties

#region Public Methods

public void SetupChart(string Title, string XaxisTitle, string Xaxis2Title, string YaxisTitle, double
XaxisMinimum, double Xaxis2Minimum, double XaxisMaximum, double Xaxis2Maximum)
{
Chart chartObj = this.FindName("XvsYChart") as Chart;

if (chartObj != null)
{
chartObj.Name = "XvsYChart";
this.Title = Title;
this.TitleXAxis = XaxisTitle;
this.TitleXAxis2 = Xaxis2Title;
this.TitleYAxis = YaxisTitle;

foreach (Series series in chartObj.Series)
{
series.ClearPoints();
```

```
}

var defaultArea = chartObj.ChartAreas[DEFAULT_CHART_AREA];
{
// Reset the zoom.
defaultArea.AxisX2.ScaleView.ZoomReset();
defaultArea.AxisY.ScaleView.ZoomReset();

defaultArea.AxisX2.Minimum = Xaxis2Minimum;
defaultArea.AxisX2.Maximum = Xaxis2Maximum;
defaultArea.AxisX2.Interval = (defaultArea.AxisX2.Maximum - defaultArea.AxisX2.Minimum) / 10;

defaultArea.AxisY.Minimum = double.NaN;
defaultArea.AxisY.Maximum = double.NaN;
defaultArea.AxisY.Interval = (defaultArea.AxisY.Maximum - defaultArea.AxisY.Minimum) / 10;

if (!defaultArea.Visible)
defaultArea.Visible = true;
}

var markerArea = chartObj.ChartAreas[Marker_CHART_AREA];
{
// Reset the zoom.
markerArea.AxisX.ScaleView.ZoomReset();
markerArea.AxisY.ScaleView.ZoomReset();

markerArea.AxisX.Minimum = XaxisMinimum;
markerArea.AxisX.Maximum = XaxisMaximum;
markerArea.AxisX.Interval = (markerArea.AxisX.Maximum - markerArea.AxisX.Minimum) / 10;

markerArea.AxisY.Minimum = 0;
markerArea.AxisY.Maximum = 1;
markerArea.AxisY.Interval = 1;

if (!markerArea.Visible)
markerArea.Visible = true;
}

}
}

public void DrawSeries(double[] xValues, double[] results, int seriesIndex, AxisType axisType)
```

```
{
var chart = this.FindName("XvsYChart") as Chart;
var series = chart.Series[seriesIndex];

series.ClearPoints();
series.Points.SuspendUpdates();
series.XAxisType = axisType;
for (int i = 0; i < xValues.Length; i++)
{
series.Points.AddXY(xValues[i], results[i]);
}
series.Points.ResumeUpdates();

enableSeries(series, true);
}

#endregion Public Methods

#region Private Methods

/// <summary>
/// Method called when the mouse is clicked. Determines if the click was on the MS Chart Legend and then checks / unchecks the Site selection.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void XvsYChart_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
{
try
{
Chart myPowerVsPoint = this.FindName("XvsYChart") as Chart;

System.Windows.Forms.DataVisualization.Charting.HitTestResult result =
myPowerVsPoint.HitTest(e.X, e.Y);

if (result != null && result.Object != null)
{
// When user hits the LegendItem
if (result.Object is LegendItem)
{
// Legend item result
LegendItem legendItem = result.Object as LegendItem;
```

```
// series item selected
Series selectedSeries = legendItem.Tag as Series;

// toggle enabled
enableSeries(selectedSeries, !selectedSeries.Enabled);

}
}
}
catch (Exception ex)
{
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}

}

private void enableSeries(Series series, bool enabled)
{
if (series.Enabled != enabled)
{
series.Enabled = enabled;
(series.Tag as LegendItem).Cells[0].Image = enabled ? IMAGE_NAME_CHECKED :
IMAGE_NAME_UNCHECKED;
}
}

#endregion Private Methods

#region Dependency Properties

public static readonly DependencyProperty ChartTypeProperty =
DependencyProperty.Register("ChartType", typeof(SeriesChartType), typeof(XvsYGenericChart),
new FrameworkPropertyMetadata
{
BindsTwoWayByDefault = true,
});

/// <summary>
/// Gets / Sets the chart type to use.
/// </summary>
public SeriesChartType ChartType
```

```
{
get
{
return (SeriesChartType)GetValue(ChartTypeProperty);
}
set
{
Chart myPowerVsPoint = this.FindName("XvsYChart") as Chart;

foreach (var series in myPowerVsPoint.Series)
{
series.ChartType = value;
//if (value == SeriesChartType.FastPoint)
//{
//    series.MarkerSize = 3;
//    series.MarkerStyle = MarkerStyle.Cross;
//}
}

SetValue(ChartTypeProperty, value);
}
}

#endregion Dependency Properties
}
}
```

.NETFramework,Version=v4.6.1.AssemblyAttributes.cs

```
// <autogenerated />
using System;
using System.Reflection;
[assembly:
global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETFramework,Version=v4.6.1",
FrameworkDisplayName = ".NET Framework 4.6.1")]
```

AssemblyInfo.cs

```
using System.Reflection;
using System.Resources;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.Windows;
```

```
// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("WaveformConverterControls")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("WaveformConverterControls")]
[assembly: AssemblyCopyright("Copyright ©  2022")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

//In order to begin building localizable applications, set
//<UICulture>CultureYouAreCodingWith</UICulture> in your .csproj file
//inside a <PropertyGroup>.  For example, if you are using US english
//in your source files, set the <UICulture> to en-US.  Then uncomment
//the NeutralResourceLanguage attribute below.  Update the "en-US" in
//the line below to match the UICulture setting in the project file.

//[assembly: NeutralResourcesLanguage("en-US", UltimateResourceFallbackLocation.Satellite)]


[assembly:ThemeInfo(
ResourceDictionaryLocation.None, //where theme specific resource dictionaries are located
//(used if a resource is not found in the page,
// or application resource dictionaries)
ResourceDictionaryLocation.SourceAssembly //where the generic resource dictionary is located
//(used if a resource is not found in the page,
// app, or any theme specific resource dictionaries)
)]


// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
```

```
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyVersion("0.14.0.01")]
[assembly: AssemblyFileVersion("0.14.0.01")]
```

Resources.Designer.cs

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace WaveformConverterControls.Properties {
using System;



/// <summary>
///   A strongly-typed resource class, for looking up localized strings, etc.
/// </summary>
// This class was auto-generated by the StronglyTypedResourceBuilder
// class via a tool like ResGen or Visual Studio.
// To add or remove a member, edit your .ResX file then rerun ResGen
// with the /str option, or rebuild your VS project.
[global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyT
ypedResourceBuilder", "16.0.0.0")]
[global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
[global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
internal class Resources {

private static global::System.Resources.ResourceManager resourceMan;

private static global::System.Globalization.CultureInfo resourceCulture;

[global::System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1811:AvoidUncalledPrivateCode")]
```

```
internal Resources() {
}

/// <summary>
///   Returns the cached ResourceManager instance used by this class.
/// </summary>
[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.Edit
orBrowsableState.Advanced)]
internal static global::System.Resources.ResourceManager ResourceManager {
get {
if (object.ReferenceEquals(resourceMan, null)) {
global::System.Resources.ResourceManager temp = new
global::System.Resources.ResourceManager("WaveformConverterControls.Properties.Resources
", typeof(Resources).Assembly);
resourceMan = temp;
}
return resourceMan;
}
}

/// <summary>
///   Overrides the current thread's CurrentUICulture property for all
///   resource lookups using this strongly typed resource class.
/// </summary>
[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.Edit
orBrowsableState.Advanced)]
internal static global::System.Globalization.CultureInfo Culture {
get {
return resourceCulture;
}
set {
resourceCulture = value;
}
}
}
}
```

Settings.Designer.cs

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
```

```
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace WaveformConverterControls.Properties {


[global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.Editors.Setti
ngsDesigner.SettingsSingleFileGenerator", "16.10.0.0")]
internal sealed partial class Settings : global::System.Configuration.ApplicationSettingsBase {

private static Settings defaultInstance =
((Settings)(global::System.Configuration.ApplicationSettingsBase.Synchronized(new Settings())));

public static Settings Default {
get {
return defaultInstance;
}
}
}
}
```

App.xaml

```
<Application x:Class="MerlinTestStudio_Demo_Telerik.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:conv="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Converters"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation">

<!--StartupUri="MainWindow.xaml"-->
<Application.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary Source="/Resources/CalResources.xaml"/>
<ResourceDictionary Source="/Resources/Styles.xaml"/>
<ResourceDictionary Source="/Resources/Templates.xaml"/>
</ResourceDictionary.MergedDictionaries>
<Style TargetType="t:RadPane" >
<Setter Property="CanDockInDocumentHost" Value="False" />
```

```
<Setter Property="Header" Value="{Binding Header}" />
<Setter Property="IsHidden" Value="{Binding IsHidden, Mode=TwoWay}" />
<Setter Property="IsActive" Value="{Binding IsActive, Mode=TwoWay}" />
</Style>
<Style TargetType="t:RadDocumentPane" >
<Setter Property="Header" Value="{Binding Header}" />
<Setter Property="IsHidden" Value="{Binding IsHidden, Mode=TwoWay}" />
<Setter Property="IsActive" Value="{Binding IsActive, Mode=TwoWay}" />
</Style>

<!--Here for backend XAML parsing of a data template for PinMap Dynamic Instrument column-->
<conv:NullToColorConverter x:Key="NullToColorConverter" />
</ResourceDictionary>
</Application.Resources>
</Application>
```

MainWindow.xaml

```
<Window x:Class="MerlinTestStudio_Demo_Telerik.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.ViewModels"
xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
xmlns:helpers="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
xmlns:localGUI="clr-namespace:MT.TestStudio.GUI"
Title="{Binding WindowTitle}" Height="450" Width="800" WindowState="Maximized"
Closing="Window_Closing">

<Window.InputBindings>
<KeyBinding Key="Z" Modifiers="Control" Command="{Binding DocumentUndoCommand}"/>
<KeyBinding Key="Y" Modifiers="Control" Command="{Binding DocumentRedoCommand}"/>
<KeyBinding Key="N" Modifiers="Control" Command="{Binding NewProjectFileCommand}"/>
<KeyBinding Key="O" Modifiers="Control" Command="{Binding OpenSolutionFileCommand}"/>
<KeyBinding Key="S" Modifiers="Control" Command="{Binding
SaveCurrentDocumentCommand}"/>
<KeyBinding Key="S" Modifiers="Ctrl+Shift"   Command="{Binding
SaveAllProjectDocumentsCommand}"/>
<KeyBinding Key="E" Modifiers="Control" Command="{Binding ExitProgramCommand}"/>
<KeyBinding Key="F1" Command="{Binding OpenUserGuideCommand}"/>
</Window.InputBindings>

<Window.Resources>
```

```
<!--This style prevents dashed bottoms of disabled textboxes from hanging around in the
Metadata tab of the WF Converter control -->
<Style TargetType="materialDesign:BottomDashedLineAdorner">
<Setter Property="Visibility" Value="Collapsed"/>
</Style>

<CollectionViewSource  x:Key="activeViewsSource" Source="{Binding Path=Panes}"
Filter="FilterActiveViewsSource" />
<CollectionViewSource  x:Key="toolboxesSource" Source="{Binding Path=Panes}"
Filter="FilterToolboxesSource" />
<t:InvertedBooleanConverter x:Key="InvertedBooleanConveter" />
<BooleanToVisibilityConverter x:Key="BoolToVis" />
<Style x:Key="ViewMenuItemStyle" TargetType="t:RadMenuItem" >
<Setter Property="IsCheckable" Value="True"/>
<Setter Property="StaysOpenOnClick" Value="True" />
<Setter Property="Header" Value="{Binding Header}" />
<Setter Property="IsChecked" Value="{Binding Path=IsHidden, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource InvertedBooleanConveter}}"
/>
</Style>
<Style x:Key="WindowMenuItemStyle" TargetType="t:RadMenuItem" BasedOn="{StaticResource
ViewMenuItemStyle}">
<Setter Property="IsCheckable" Value="True" />
<Setter Property="Header" Value="{Binding Path=Header}"/>
<Setter Property="IsChecked" Value="{Binding Path=IsActive, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
</Style>

<DataTemplate x:Key="TitleTemplate">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="25"/>
</Grid.ColumnDefinitions>
<ContentPresenter Content="{Binding}"   Margin="0 0 10 0"/>
<t:RadButton x:Name="RadPaneMaxMinButton"  Grid.Column="1"
Click="RadPaneMaxMinButton_Click" >
<t:RadButton.Content>
<t:RadGlyph Glyph="&#xe138;" Width="Auto" Height="Auto" Foreground="White"/>
</t:RadButton.Content>
</t:RadButton>
</Grid>
```

```
</DataTemplate>

<!--<Style TargetType="t:RadPane">
<Setter Property="TitleTemplate" Value="{StaticResource TitleTemplate}" />
</Style>-->

<!--Define some colours to be used with the Tabs.-->
<SolidColorBrush x:Key="WindowTitleBar" Color="#252526" />
<SolidColorBrush x:Key="WindowTitleBarButtonHover" Color="#3F3F41" />


<!--Style the Windows Title Bar.-->
<Style x:Key="CustomWindowStyle" TargetType="{x:Type Window}">
<Setter Property="WindowChrome.WindowChrome">
<Setter.Value>
<WindowChrome CaptionHeight="30"
CornerRadius="0"
GlassFrameThickness="0"
NonClientFrameEdges="None"
ResizeBorderThickness="5"
UseAeroCaptionButtons="False" />
</Setter.Value>
</Setter>
<Setter Property="BorderBrush" Value="{StaticResource WindowTitleBar}" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Window}">
<Grid>
<Border Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="5,35,5,5">
<AdornerDecorator>
<ContentPresenter />
</AdornerDecorator>
</Border>

<DockPanel Height="30"
VerticalAlignment="Top"
LastChildFill="False" Margin="8,4,0,0">

<!--<Image Margin="3" Source="{TemplateBinding Icon}" />-->
<Image Margin="3,3,3,3" Source="Resources\Images\Branding\AppIcon.ico" />
```

```
<TextBlock Margin="10,4,0,0"
VerticalAlignment="Center"
DockPanel.Dock="Left"
FontSize="12"
Foreground="White"
Text="{TemplateBinding Title}" />

<!--Close Button.-->
<Button DockPanel.Dock="Right"
HorizontalAlignment="Right"
Click="CloseClick"
WindowChrome.IsHitTestVisibleInChrome="True">
<Button.Style>
<Style TargetType="{x:Type Button}">
<Setter Property="Background" Value="{StaticResource WindowTitleBar}" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
<Border Background="{TemplateBinding Background}">
<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="BorderBrush" Value="{StaticResource WindowTitleBar}" />
<Setter Property="Foreground" Value="Pink" />
<Setter Property="Margin" Value="0,2,8,0" />
<Setter Property="Height" Value="30" />
<Setter Property="Width" Value="30" />
<Style.Triggers>
<Trigger Property="IsMouseOver" Value="True">
<Setter Property="Background" Value="{StaticResource WindowTitleBarButtonHover}"/>
</Trigger>
</Style.Triggers>
</Style>

</Button.Style>
<Button.ToolTip>
<StackPanel>
<TextBlock Text="Close" />
</StackPanel>
```

```
</Button.ToolTip>
<StackPanel Orientation="Horizontal">
<Image Height="16" Source="Resources\Images\WindowsTitleBar\window-close.png" />
</StackPanel>
</Button>

<!--Restore Button.-->
<Button DockPanel.Dock="Right"
HorizontalAlignment="Right"
Click="MaximizeRestoreClick"
WindowChrome.IsHitTestVisibleInChrome="True">
<Button.Style>
<Style TargetType="{x:Type Button}">
<Setter Property="Background" Value="{StaticResource WindowTitleBar}" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
<Border Background="{TemplateBinding Background}">
<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="BorderBrush" Value="{StaticResource WindowTitleBar}" />
<Setter Property="Foreground" Value="Pink" />
<Setter Property="Margin" Value="0,0,0,0" />
<Setter Property="Height" Value="30" />
<Setter Property="Width" Value="40" />
<Style.Triggers>
<Trigger Property="IsMouseOver" Value="True">
<Setter Property="Background" Value="{StaticResource WindowTitleBarButtonHover}"/>
</Trigger>
</Style.Triggers>
</Style>

</Button.Style>
<Button.ToolTip>
<StackPanel>
<TextBlock Text="Restore" />
</StackPanel>
</Button.ToolTip>
<StackPanel Orientation="Horizontal">
```

```
<Image Height="16" Source="Resources\Images\WindowsTitleBar\window-restore.png" />
</StackPanel>
</Button>

<!--Minimize Button-->
<Button DockPanel.Dock="Right"
HorizontalAlignment="Right"
Click="MinimizeClick"
WindowChrome.IsHitTestVisibleInChrome="True">
<Button.Style>
<Style TargetType="{x:Type Button}">
<Setter Property="Background" Value="{StaticResource WindowTitleBar}" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
<Border Background="{TemplateBinding Background}">
<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="BorderBrush" Value="{StaticResource WindowTitleBar}" />
<Setter Property="Foreground" Value="Pink" />
<Setter Property="Margin" Value="0,2,0,0" />
<Setter Property="Height" Value="30" />
<Setter Property="Width" Value="30" />
<Style.Triggers>
<Trigger Property="IsMouseOver" Value="True">
<Setter Property="Background" Value="{StaticResource WindowTitleBarButtonHover}"/>
</Trigger>
</Style.Triggers>
</Style>

</Button.Style>
<Button.ToolTip>
<StackPanel>
<TextBlock Text="Restore" />
</StackPanel>
</Button.ToolTip>
<StackPanel Orientation="Horizontal">
<Image Height="16" Source="Resources\Images\WindowsTitleBar\window-minimize.png" />
</StackPanel>
```

```
</Button>

</DockPanel>

</Grid>

</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

</Window.Resources>

<t:RadBusyIndicator x:Name="mBusyIndicator" BusyContent="Loading Content"  IsBusy="False"
IsIndeterminate="True">
<Grid>
<!--Grid Row/Column Definitions-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="30"/>
<RowDefinition Height="*"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<!--Main Menu-->
<t:RadMenu Grid.Row="0">
<t:RadMenu.ItemContainerStyle>
<Style TargetType="t:RadMenuItem">
<Setter Property="Margin" Value="0 0 0 0"/>
<Setter Property="Padding" Value="5"/>
</Style>
</t:RadMenu.ItemContainerStyle>
<!--File MenuItem-->
<t:RadMenuItem Header="File"  >

<t:RadMenuItem Header="New Project" Command="{Binding Path=NewProjectFileCommand}"
InputGestureText="Ctrl+N">
<t:RadMenuItem.Icon>
<Image Source="Images\Custom-Icon-Design-Flatastic-10-New-file.ico" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
```

```
<t:RadMenuItem Header="Open Existing Project" Command="{Binding
Path=OpenSolutionFileCommand}" InputGestureText="Ctrl+O">
<t:RadMenuItem.Icon>
<Image Source="Images\SearchFolderOpened_16x.png" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>

<t:RadMenuSeparatorItem/>

<t:RadMenuItem Header="Save File" InputGestureText="Ctrl+S" Command="{Binding
SaveCurrentDocumentCommand}">
<t:RadMenuItem.Icon>
<Image Source="Images\Save_16x.png" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Header="Save All" InputGestureText="Ctrl+Shift+S" Command="{Binding
SaveAllProjectDocumentsCommand}">
<t:RadMenuItem.Icon>
<Image Source="Images\SaveAll_16x.png" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>

<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Recent Projects" ItemsSource="{Binding
Path=MyRecentData.RecentProjects, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
<t:RadMenuItem.ItemContainerStyle>
<Style TargetType="t:RadMenuItem">
<Setter Property="Header" Value="{Binding
Path=GetRecentDisplayHeader,Mode=OneWay,UpdateSourceTrigger=PropertyChanged}"/>
<Setter Property="Command" Value="{Binding RelativeSource={RelativeSource FindAncestor,
AncestorType={x:Type t:RadMenu}}, Path=DataContext.OpenRecentMerlinProjectCommand}" />
<Setter Property="CommandParameter" Value="{Binding Path=ProjectFilePath}"/>
</Style>
</t:RadMenuItem.ItemContainerStyle>
</t:RadMenuItem>

<!--<t:RadMenuItem Header="Save" x:Name="SaveMenuItem" Command="{Binding
Path=MainSave}"/>
<t:RadMenuItem Header="Load" x:Name="LoadMenuItem" Command="{Binding
Path=MainLoad}"/>-->
<t:RadMenuSeparatorItem/>
```

```
<t:RadMenuItem Header="Preferences" Command="{Binding
Path=OpenPreferencesCommand}">
<t:RadMenuItem.Icon>
<t:RadGlyph Glyph="&#xe13a;"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Exit" Command="{Binding ExitProgramCommand}"
InputGestureText="Ctrl+E">
<t:RadMenuItem.Icon>
<Image Source="Images\305_Close_48x48_72.png" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadMenuItem>
<!--Edit MenuItem-->
<t:RadMenuItem Header="Edit"  >
<t:RadMenuItem Header="Undo" Command="{Binding DocumentUndoCommand}" />
<t:RadMenuItem Header="Redo"  Command="{Binding DocumentRedoCommand}"/>
</t:RadMenuItem>
<!--View MenuItem-->
<t:RadMenuItem Header="View"
ItemContainerStyle="{StaticResource ViewMenuItemStyle}"
ItemsSource="{Binding Source={StaticResource toolboxesSource}}">
</t:RadMenuItem>

<t:RadMenuItem Header="Project">
<t:RadMenuItem Header="Compile DUT Sequence" Padding="2" Command="{Binding
Path=CompileCommand}">
<t:RadMenuItem.Icon>
<t:RadGlyph Glyph="&#xe132;" Foreground="White"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadMenuItem>
<!--Tools MenuItem-->
<t:RadMenuItem Header="Tools"  >
<t:RadMenuItem Header="RF110 SFP" Padding="2" Margin="0 0 3 0" Command="{Binding
Path=OpenRF110_SFP_Command}">
<t:RadMenuItem.Icon>
<t:RadGlyph Glyph="&#xe91b;" Foreground="White"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Header="RF210 SFP" Padding="2" Margin="0 0 3 0" Command="{Binding
```

```
Path=OpenRF210_SFP_Command}">
<t:RadMenuItem.Icon>
<t:RadGlyph Glyph="&#xe91b;" Foreground="White"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadMenuItem>
<t:RadMenuItem Header="Window"
x:Name="WindowRadMenu"
ItemsSource="{Binding Source={StaticResource activeViewsSource}}"
ItemContainerStyle="{StaticResource WindowMenuItemStyle}">
</t:RadMenuItem>
<!--Help MenuItem-->
<t:RadMenuItem Header="Help" >
<t:RadMenuItem Header="About" Command="{Binding OpenAboutBoxCommand}" />
<t:RadMenuItem Header="Theme" Visibility="Collapsed">
<t:RadMenuItem Header="Dark" Command="{Binding ChangeThemeCommand}"
CommandParameter="Dark"/>
<t:RadMenuItem Header="Light" Command="{Binding ChangeThemeCommand}"
CommandParameter="Light"/>
<t:RadMenuItem Header="Blue" Command="{Binding ChangeThemeCommand}"
CommandParameter="Blue"/>
</t:RadMenuItem>
</t:RadMenuItem>

</t:RadMenu>

<!--Define a tool bar to run along the top of the application.-->
<ToolBarTray DockPanel.Dock="Top" IsLocked="True" Background="#2D2D30" Grid.Row="1">

<ToolBar Background="#2D2D30" ClipToBounds="False" HorizontalAlignment="Left"
Width="3000"  Margin="5,-2,0,0">
<Separator />
<Button Height="20" Width="20" Padding="0" Margin="3"
Command="{Binding NewProjectFileCommand}"
ToolTip="New Merlin Test Studio Project File (Ctrl-N)">
<Image Source="Images\Custom-Icon-Design-Flatastic-10-New-file.ico" />
</Button>

<Button Height="20" Width="20"  Padding="0"  Margin="3"
Command="{Binding OpenSolutionFileCommand}"
ToolTip="Open Merlin Test Studio Project File (Ctrl-O)">
<Image Source="Images\SearchFolderOpened_16x.png" />
```

```
</Button>

<Button Height="20" Width="20"  Padding="0"      Margin="3"
Command="{Binding SaveCurrentDocumentCommand}"
ToolTip="Save Current File (Ctrl-S)">
<Image Source="Images\Save_16x.png" />
</Button>

<Button Height="20" Width="20"  Padding="0"  Margin="3"
Command="{Binding SaveAllProjectDocumentsCommand}"
ToolTip="Save All (Ctlr-Shift-S)">
<Image Source="Images\SaveAll_16x.png" />
</Button>
<Separator />
<Button Height="20" Width="20"  Padding="0"  Margin="3"
Command="{Binding DocumentUndoCommand}"
ToolTip="Undo (Ctlr-Z)">
<Image Source="Images\Undo_16x.png" />
</Button>
<Button Height="20" Width="20"  Padding="0"  Margin="3"
Command="{Binding DocumentRedoCommand}"
ToolTip="Redo (Ctlr-Y)">
<Image Source="Images\Redo_16x.png" />
</Button>
<Separator />
<t:RadToggleSwitchButton x:Name="DevModeEzAccessToggle" Margin="5"
HorizontalAlignment="Left" VerticalAlignment="Center"
UncheckedContent="User" CheckedContent="DevMode"
IsChecked="{Binding Path=Preferences.IsDevModeActive, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
/>
<Separator/>
<t:RadButton BorderThickness="0" Command="{Binding BringPaneIntoViewCommand}">
<t:RadButton.Content>
<Grid HorizontalAlignment="Center" VerticalAlignment="Center">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="20"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<t:RadGlyph Grid.Column="0" FontSize="14" HorizontalAlignment="Center"
VerticalAlignment="Center">
<t:RadGlyph.Style>
```

```
<Style TargetType="t:RadGlyph">
<Setter Property="Glyph" Value="&#xe11d;"/>
<Setter Property="Foreground" Value="IndianRed"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=ErrorLog.Count, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="0">
<Setter Property="Glyph" Value="&#xe11a;"/>
<Setter Property="Foreground" Value="ForestGreen"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadGlyph.Style>
</t:RadGlyph>
<TextBlock Grid.Column="1" FontSize="12" Foreground="White" HorizontalAlignment="Center"
VerticalAlignment="Center" >
<TextBlock.Style>
<Style TargetType="TextBlock">
<Setter Property="Text" Value="{Binding Path=ErrorLog.Count, StringFormat={}0 Errors,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=ErrorLog.Count, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="1">
<Setter Property="Text" Value="{Binding Path=ErrorLog.Count, StringFormat={}0 Error,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"/>
</DataTrigger>
<DataTrigger Binding="{Binding Path=ErrorLog.Count, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="0">
<Setter Property="Text" Value="No issues found"/>
</DataTrigger>
</Style.Triggers>
</Style>
</TextBlock.Style>
</TextBlock>

</Grid>
</t:RadButton.Content>
</t:RadButton>
</ToolBar>
</ToolBarTray>


<!--Main Workspace--> <!--Source => Path=Panes-->
```

```
<t:RadDocking x:Name="mainDocking" Grid.Row="2"
PreviewShowCompass="mainDocking_PreviewShowCompass"
PanesSource="{Binding Path=Panes}"
Close="mainDocking_Close"
RetainPaneSizeMode="DockingAndFloating"
CloseButtonPosition="InPane" >
<t:RadDocking.Resources>

</t:RadDocking.Resources>
<t:RadDocking.DockingPanesFactory>
<helpers:CustomDockingPanesFactory />
</t:RadDocking.DockingPanesFactory>
<t:RadDocking.CurrentSaveLoadLayoutHelper>
<helpers:CustomSaveLoadLayoutHelper />
</t:RadDocking.CurrentSaveLoadLayoutHelper>

<t:RadSplitContainer MaxWidth="600" InitialPosition="DockedLeft">
<t:RadPaneGroup Name="leftGroup" t:RadDocking.SerializationTag="leftGroup">
</t:RadPaneGroup>
</t:RadSplitContainer>

<t:RadSplitContainer MaxWidth="679" InitialPosition="DockedRight">
<t:RadPaneGroup x:Name="rightGroup" t:RadDocking.SerializationTag="rightGroup">
</t:RadPaneGroup>
</t:RadSplitContainer>

<t:RadSplitContainer InitialPosition="DockedBottom" MaxHeight="5in">
<t:RadPaneGroup x:Name="bottomGroup" t:RadDocking.SerializationTag="bottomGroup">
</t:RadPaneGroup>
</t:RadSplitContainer>

</t:RadDocking>

<!--A statusBar is used tp provide basic feedback to the user.-->
<StatusBar x:Name="TestStatusBar" DockPanel.Dock="Bottom" Background="{Binding
NonTelerikControlBackGroundColor}" Grid.Row="3">

<StatusBarItem DockPanel.Dock="Left" Width="800">
<DockPanel LastChildFill="True">
<!--<Label Content="" Grid.Row="0" Width="55" ToolTip=""/>-->
<Label Foreground="White" Content="{Binding Status, Mode=OneWay}" Grid.Row="0"
ToolTip="Status of Merlin Test Studio" Margin="5,3,0,0"/>
```

```
</DockPanel>
</StatusBarItem>

<StatusBarItem HorizontalContentAlignment="Stretch" Margin="5,0,5,5">
<!-- Note extra attribute -->
<Grid Margin="0" Grid.Row="0" Visibility="{Binding ProgressBarVisibility}">
<!--<ProgressBar Margin="10,0,10,0" Height="20" IsIndeterminate="True"
Foreground="CornflowerBlue" Visibility="{Binding ProgressBarVisibility}" Minimum="0"
Maximum="100"/>-->
<ProgressBar Height="20" Margin="0,5,0,0" Minimum="0" Maximum="100" Value="{Binding
PercentageComplete}" Name="pbStatus" Style="{x:Null}" />
<TextBlock Margin="0,5,0,0" Text="{Binding ElementName=pbStatus, Path=Value,
StringFormat={}{0:0}%}" HorizontalAlignment="Center" VerticalAlignment="Center"
Foreground="Black" />
</Grid>
</StatusBarItem>



</StatusBar>

</Grid>
</t:RadBusyIndicator>
</Window>
```

CalResources.xaml

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation">

<Style x:Key="MenuItemContainerStyle" TargetType="t:RadMenuItem">
<Setter Property="Header" Value="{Binding Path=Text}"          />
<Setter Property="ItemsSource" Value="{Binding Path=SubItems}"/>
<Setter Property="IsEnabled" Value="{Binding Path=IsEnabled}"/>
</Style>
<Style x:Key="BetterStyle" TargetType="t:RadListBoxItem">
<Setter Property="t:DragDropManager.AllowCapturedDrag" Value="True"/>
</Style>

<!--Fine Grain data column style in Attenuator Results-->
<Style x:Key="RegressionDataColumnStyle" TargetType="t:GridViewDataColumn">
<Setter Property="IsVisible" Value="False"/>
```

```
<Style.Triggers>
<DataTrigger Binding="{Binding Path=ShowPolynomialRegressionControls, Mode=OneWay}"
Value="True">
<Setter Property="IsVisible" Value="True"/>
</DataTrigger>
</Style.Triggers>
</Style>
<!--Course Grain data column style in Attenuator Results-->
<Style x:Key="CourseDataColumnStyle" TargetType="t:GridViewDataColumn">
<Setter Property="IsVisible" Value="True"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=ShowPolynomialRegressionControls, Mode=OneWay}"
Value="True">
<Setter Property="IsVisible" Value="False"/>
</DataTrigger>
</Style.Triggers>
</Style>

<Style x:Key="ProductInfoLabelStyle" TargetType="TextBlock">
<Setter Property="Foreground" Value="White"/>
<Setter Property="FontSize" Value="12"/>
<Setter Property="HorizontalAlignment" Value="Right"/>
<Setter Property="VerticalAlignment" Value="Center"/>
<Setter Property="Margin" Value="5 5 0 5"/>
</Style>
<Style x:Key="ProductInfoInputStyle" TargetType="TextBox">
<Setter Property="FontSize" Value="12"/>
<Setter Property="Margin" Value="0 5 5 5"/>
</Style>
<!--Check Box Style for AmpConfigs-->
<Style x:Key="CheckBoxStyle" TargetType="CheckBox">
<Setter Property="Background" Value="IndianRed"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=IsChecked, RelativeSource={ RelativeSource Self} }"
Value="True">
<Setter Property="Background" Value="ForestGreen"/>
</DataTrigger>
</Style.Triggers>
</Style>

<!-- ComboBoxDropDown on Cell edit Style -->
<Style x:Key="DropDownOnCellEdit" TargetType="t:RadComboBox">
```

```
<Setter Property="OpenDropDownOnFocus" Value="True"/>
</Style>
<!--Cal Point Invalid/Valid Row Style-->
<Style TargetType="t:GridViewRow" x:Key="CalPointGridViewRowStyle">

<Style.Triggers>
<DataTrigger Binding="{Binding Path=PointStatus, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="2">
<Setter Property="BorderBrush" Value="Red"/>
<Setter Property="BorderThickness" Value="1"/>
<Setter Property="Background" Value="#40282d"/>
<Setter Property="SelectedBackground" Value="#cf4e4e"/>
<Setter Property="ToolTip" Value="{Binding Path=ToolTipString, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"/>
</DataTrigger>
</Style.Triggers>
</Style>
<!--Warning Cell style-->
<Style x:Key="WarningGridViewCellStyle" TargetType="t:GridViewCell">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=HasWarning, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="true">
<Setter Property="Foreground" Value="Orange"></Setter>
<Setter Property="ToolTip" Value="{Binding Path=WarningMessage, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" />
</DataTrigger>
</Style.Triggers>

</Style>

<!--AmplifiersCompressed Cell style-->
<Style x:Key="AmplifiersCompressedCellStyle" TargetType="t:GridViewCell">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=Value, RelativeSource={ RelativeSource Self} }"
Value="Good">

<Setter Property="SelectedBackground" Value="DodgerBlue"/>
</DataTrigger>
<DataTrigger Binding="{Binding Path=Value, RelativeSource={ RelativeSource Self} }"
Value="Compressed">

<Setter Property="SelectedBackground" Value="DodgerBlue"/>
```

```
</DataTrigger>
</Style.Triggers>
</Style>

<!--Cal Step Cell style-->
<Style x:Key="CalStepCellStyle" TargetType="t:GridViewCell">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=Value, RelativeSource={ RelativeSource Self} }"
Value="Success">

</DataTrigger>
<DataTrigger Binding="{Binding Path=Value, RelativeSource={ RelativeSource Self} }" Value="-">

</DataTrigger>
</Style.Triggers>
</Style>

<!--Pass/Fail Cell style-->
<Style x:Key="PassFailCellStyle" TargetType="t:GridViewCell">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=Value, RelativeSource={ RelativeSource Self} }"
Value="true">
<Setter Property="Background" Value="#314f2f"></Setter>
</DataTrigger>
<DataTrigger Binding="{Binding Path=Value, RelativeSource={ RelativeSource Self} }"
Value="false">
<Setter Property="Background" Value="#40282d"></Setter>
</DataTrigger>
</Style.Triggers>
</Style>

</ResourceDictionary>
```

Styles.xaml

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation">

<SolidColorBrush x:Key="VS_Black" Color="#FF252526" />
<SolidColorBrush x:Key="VS_Red" Color="#FFE51400" />
<SolidColorBrush x:Key="VS_Blue" Color="#FF007ACC" />
```

```
<!-- The Toggle Button style -->
<Style x:Key="ToggleButtonStyle" TargetType="t:RadToggleButton">
<Setter Property="HorizontalContentAlignment" Value="Right"/>
<Setter Property="BorderThickness" Value="1"/>
<Setter Property="Padding" Value="8 1"/>
<Setter Property="FocusVisualStyle" Value="{x:Null}"/>
<Setter Property="FontSize" Value="12"/>
<Setter Property="FontFamily" Value="Segoe UI"/>
<Setter Property="Height" Value="24"/>
<Setter Property="Width" Value="61"/>
<Setter Property="VerticalContentAlignment" Value="Center"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="t:RadToggleButton">
<Grid>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="CheckStates">
<VisualStateGroup.Transitions>
<VisualTransition From="Unchecked"  To="Checked">
<Storyboard>
<DoubleAnimationUsingKeyFrames
Storyboard.TargetProperty="(Ellipse.RenderTransform).(TranslateTransform.X)"
Storyboard.TargetName="Thumb">
<EasingDoubleKeyFrame KeyTime="0" Value="0"/>
<EasingDoubleKeyFrame KeyTime="0:0:0.4" Value="38">
<EasingDoubleKeyFrame.EasingFunction>
<SineEase EasingMode="EaseOut"/>
</EasingDoubleKeyFrame.EasingFunction>
</EasingDoubleKeyFrame>
</DoubleAnimationUsingKeyFrames>

<DoubleAnimation Duration="0:0:0.6" Storyboard.TargetName="Content"
Storyboard.TargetProperty="Opacity" To="1" From="0" />
</Storyboard>
</VisualTransition>
<VisualTransition From="Checked"  To="UnChecked">
<Storyboard>
<DoubleAnimationUsingKeyFrames
Storyboard.TargetProperty="(Ellipse.RenderTransform).(TranslateTransform.X)"
Storyboard.TargetName="Thumb">
<EasingDoubleKeyFrame KeyTime="0" Value="38"/>
<EasingDoubleKeyFrame KeyTime="0:0:0.3" Value="0">
```

```
<EasingDoubleKeyFrame.EasingFunction>
<SineEase EasingMode="EaseOut"/>
</EasingDoubleKeyFrame.EasingFunction>
</EasingDoubleKeyFrame>
</DoubleAnimationUsingKeyFrames>

<DoubleAnimation Duration="0:0:0.6" Storyboard.TargetName="Content"
Storyboard.TargetProperty="Opacity" To="1" From="0" />
</Storyboard>
</VisualTransition>
</VisualStateGroup.Transitions>

<VisualState x:Name="Checked">
<Storyboard>
<DoubleAnimationUsingKeyFrames
Storyboard.TargetProperty="(Ellipse.RenderTransform).(TranslateTransform.X)"
Storyboard.TargetName="Thumb">
<EasingDoubleKeyFrame KeyTime="0" Value="38"/>
</DoubleAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Unchecked">
<Storyboard>
<DoubleAnimationUsingKeyFrames
Storyboard.TargetProperty="(Ellipse.RenderTransform).(TranslateTransform.X)"
Storyboard.TargetName="Thumb">
<EasingDoubleKeyFrame KeyTime="0" Value="0"/>
</DoubleAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Grid SnapsToDevicePixels="True">
<Rectangle x:Name="Track" Fill="{t:VisualStudio2013Resource ResourceKey=BasicBrush}"
Stroke="{x:Null}" RadiusY="13" RadiusX="13" />

<Ellipse x:Name="Thumb" Fill="{t:VisualStudio2013Resource ResourceKey=PrimaryBrush}"
HorizontalAlignment="Left" StrokeThickness="1"
Stroke="{t:VisualStudio2013Resource ResourceKey=BasicBrush}" VerticalAlignment="Top"
Width="22" Height="22" RenderTransformOrigin="0.5,0.5">
<Ellipse.RenderTransform>
<TranslateTransform X="0" Y="0" />
```

```
</Ellipse.RenderTransform>
<Ellipse.Effect>
<DropShadowEffect BlurRadius="5" ShadowDepth="1" Direction="270" Color="#CCCCCC"/>
</Ellipse.Effect>
</Ellipse>
</Grid>

<ContentPresenter x:Name="Content"
Margin="{TemplateBinding Padding}"
Content="{TemplateBinding Content}"
ContentTemplate="{TemplateBinding ContentTemplate}"
VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
ContentTemplateSelector="{TemplateBinding ContentTemplateSelector}"
ContentStringFormat="{TemplateBinding ContentStringFormat}"
RecognizesAccessKey="True"/>
</Grid>
<ControlTemplate.Triggers>
<Trigger Property="IsMouseOver" Value="True">
<Setter TargetName="Thumb" Property="Fill" Value="{t:VisualStudio2013Resource
ResourceKey=MouseOverBrush}" />
<Setter TargetName="Thumb" Property="Stroke" Value="{t:VisualStudio2013Resource
ResourceKey=AccentBrush}" />
</Trigger>
<Trigger Property="IsChecked" Value="True">
<Setter TargetName="Thumb" Property="Fill" Value="{t:VisualStudio2013Resource
ResourceKey=AccentBrush}" />
<Setter TargetName="Thumb" Property="Stroke" Value="{t:VisualStudio2013Resource
ResourceKey=AccentBrush}" />
<Setter TargetName="Track" Property="Opacity" Value="0.4" />
<Setter TargetName="Track" Property="Fill" Value="{t:VisualStudio2013Resource
ResourceKey=AccentBrush}" />

<Setter TargetName="Content" Property="HorizontalAlignment" Value="Left" />
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!--ListBox Headers-->
```

```
<Style x:Key="FuctionSelectionListBoxHeader" TargetType="TextBlock">
<Setter Property="HorizontalAlignment" Value="Center"/>
<Setter Property="VerticalAlignment" Value="Center" />
<Setter Property="FontSize" Value="20"            />
<Setter Property="Width" Value="Auto"            />
<Setter Property="Height" Value="Auto"          />
<Setter Property="Foreground" Value="LightGray" />
</Style>

<!--Parameter Headers-->
<Style x:Key="ParameterGridHeaders" TargetType="TextBlock">
<Setter Property="HorizontalAlignment" Value="Left" />
<Setter Property="VerticalAlignment" Value="Center"/>
<Setter Property="FontSize" Value="16"           />
<Setter Property="Width" Value="Auto"           />
<Setter Property="Height" Value="Auto"         />
<Setter Property="Foreground" Value="LightGray"/>
</Style>

<!--ListBox-->
<Style x:Key="NormListBoxItem" TargetType="t:RadListBoxItem">
<Setter Property="FontSize" Value="14"/>
<Setter Property="t:DragDropManager.AllowCapturedDrag" Value="True" />
</Style>
<Style x:Key="DraggableListBoxItem" TargetType="t:RadListBoxItem" BasedOn="{StaticResource
NormListBoxItem}">
<Setter Property="Content" Value="{Binding Path=FunctionName}"/>
</Style>

<!-- ComboBoxDropDown on Cell edit Style -->
<Style x:Key="DropDownOnCellEdit" TargetType="t:RadComboBox">
<Setter Property="OpenDropDownOnFocus" Value="False"/>
</Style>
<Style x:Key="NullableEnumRadComboBoxStyle" TargetType="t:RadComboBox"
BasedOn="{StaticResource DropDownOnCellEdit}">
<Setter Property="ClearSelectionButtonVisibility" Value="Visible" />
<Setter Property="ClearSelectionButtonContent" Value="Clear" />
</Style>

<!--DragAndDropRow default Style-->
<Style TargetType="t:GridViewRow" x:Key="GridViewRowDragDropBase">
<Setter Property="t:DragDropManager.AllowDrag"
```

```
Value="True" />
<Setter Property="t:DragDropManager.TouchDragTrigger"
Value="TapAndHold"/>
</Style>

<!--Command Based Row Styles-->
<Style TargetType="t:GridViewRow" x:Key="NormalRowStyle" BasedOn="{StaticResource
GridViewRowDragDropBase}"/> <!--NORMAL-->
<Style TargetType="t:GridViewRow" x:Key="HaltStyle" > <!--HALT-->
<Setter Property="Background" Value="Red"/>
</Style>
<Style TargetType="t:GridViewRow" x:Key="EnabledStyle" BasedOn="{StaticResource
NormalRowStyle}"> <!--ENABLED ROW-->
<Setter Property="IsEnabled" Value="True"/>
</Style>
<Style TargetType="t:GridViewRow" x:Key="DisabledStyle" > <!--DISABLED ROW-->
<Setter Property="IsEnabled" Value="False"/>
</Style>

<!--Column Group Header Style for PinMap-->
<Style x:Key="ColumnGroupHeaderStyle" TargetType="t:CommonColumnHeader" >
<Setter Property="FontSize" Value="14"/>
<Setter Property="TextBlock.VerticalAlignment" Value="Center"/>
</Style>
<Style TargetType="t:GridViewRow" x:Key="PinMapGridViewRowStyle">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=Status, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="2">
<Setter Property="BorderBrush" Value="Red"/>
<Setter Property="BorderThickness" Value="1"/>
<Setter Property="Background" Value="#40282d"/>
<Setter Property="SelectedBackground" Value="#cf4e4e"/>
<Setter Property="ToolTip" Value="{Binding Path=ToolTipString, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"/>
</DataTrigger>
</Style.Triggers>
</Style>

<!--Test Styles-->
<Style x:Key="TestBaseStyle" TargetType="t:GridViewRow" BasedOn="{StaticResource
GridViewRowDragDropBase}">
```

```
</Style>
<Style x:Key="TestBreakStyle" TargetType="t:GridViewRow" BasedOn="{StaticResource
GridViewRowDragDropBase}">
<Setter Property="Background" Value="DarkGreen" />
</Style>

<!-- Context menu item style-->
<Style x:Key="ContextMenuItemStyle" TargetType="t:RadMenuItem">
<Setter Property="Icon" Value="{Binding IconUrl}"/>
<Setter Property="IconTemplate">
<Setter.Value>
<DataTemplate>
<Image Source="{Binding}" Stretch="None"/>
</DataTemplate>
</Setter.Value>
</Setter>
<Setter Property="IsSeparator" Value="{Binding IsSeparator}"/>
<Setter Property="Header" Value="{Binding Text}"/>
<Setter Property="ItemsSource" Value="{Binding SubItems}"/>
<Setter Property="Command" Value="{Binding Command}"/>
<!-- ??? Need command parameter value binding ???-->
</Style>

<!--Test Limits Column Style Temp-->
<Style x:Key="TestLimits_ColumnStyle" TargetType="t:GridViewColumn">
<Setter Property="HeaderTextAlignment" Value="Center"/>
<Setter Property="HeaderTextWrapping" Value="Wrap"/>
<Setter Property="TextAlignment" Value="Center"/>
</Style>

</ResourceDictionary>
```

Templates.xaml

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
xmlns:PointerEnumLocal="clr-namespace:MerlinTestStudio_Demo_Telerik.Data">



<Style x:Key="CenteredCellStyle" TargetType="t:GridViewCell">
<Setter Property="HorizontalContentAlignment" Value="Center"/>
```

```
</Style>

<DataTemplate x:Key="FunctionItemTemplate" >
<Border ToolTipService.ToolTip="{Binding Path=DLLClassType}">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<TextBlock Text="{Binding Path=FunctionName}" FontSize="14" HorizontalAlignment="Center"
Grid.Column="0"              />
<Border HorizontalAlignment="Right" CornerRadius="6" Margin="20 0 0 0"
Background="{StaticResource VS_Blue}">
<TextBlock Text=" + " FontSize="14" />
</Border>
</Grid>
</Border>
</DataTemplate>

<DataTemplate x:Key="SequenceControlTemplate" >
<Grid>
<TextBlock Text="{Binding Path=ControlName}" FontSize="14" HorizontalAlignment="Left"
Grid.Column="0"/>
</Grid>
</DataTemplate>



<!-- Parameter Pointer Template -->
<DataTemplate x:Key="ParameterPointerTemplate">
<Border BorderBrush="DimGray" BorderThickness="0">
<Grid Margin="2">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="2*"  />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="0.5*"/>
</Grid.ColumnDefinitions>
<TextBlock Text="{Binding Path=ParamType}" FontSize="14" Grid.Column="0"/>
<TextBlock Text="{Binding Path=ParamName}" FontSize="14" Grid.Column="1"/>

<TextBox Text="{Binding Path=TempParamMappingValue, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" FontSize="14" Width="Auto" Grid.Column="2" >
```

```
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="TextChanged"
Command="{Binding Path=DataContext.TempParamValueChanged,
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type t:RadListBox}}}"
CommandParameter="{Binding Path=Content, RelativeSource={RelativeSource
Mode=FindAncestor, AncestorType={x:Type t:RadListBoxItem}}}">
</t:EventBinding>
</t:EventToCommandBehavior.EventBindings>
<TextBox.Style>
<Style TargetType="TextBox">
<Setter Property="Visibility" Value="Visible"/>
<Setter Property="IsEnabled" Value="{Binding Path=IsMapped}"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=IsEnum}" Value="true">
<Setter Property="Visibility" Value="Collapsed"/>
</DataTrigger>
</Style.Triggers>
</Style>
</TextBox.Style>
</TextBox>

<t:RadComboBox Width="Auto" Grid.Column="2" x:Name="ParamColumnBox"
SelectedItem="{Binding Path=TempParamMappingValue, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=ParamOptions}"
ClearSelectionButtonVisibility="Visible"
ClearSelectionButtonContent="No Enum">
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="MouseLeftButtonUp" Command="{Binding
Path=DataContext.TempParamValueChanged, RelativeSource={RelativeSource
Mode=FindAncestor, AncestorType={x:Type t:RadListBox}}}"
RaiseOnHandledEvents="True" CommandParameter="{Binding Path=Content,
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type
t:RadListBoxItem}}}"/>
</t:EventToCommandBehavior.EventBindings>
<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Setter Property="Visibility" Value="Collapsed"/>
<Setter Property="IsEnabled" Value="{Binding Path=IsMapped}"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=IsEnum}" Value="true">
<Setter Property="Visibility" Value="Visible"/>
```

```
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>

<t:RadComboBox Width="Auto" Grid.Column="3" x:Name="UnitColumnBox"
IsEnabled="{Binding Path=ParamMapping.IsUnitable, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedValue="{Binding Path=ParamUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=ParamMapping.Units, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged }">
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="MouseLeftButtonUp" Command="{Binding
Path=DataContext.ReadTempParamValue, RelativeSource={RelativeSource Mode=FindAncestor,
AncestorType={x:Type t:RadListBox}}}"
RaiseOnHandledEvents="True" CommandParameter="{Binding Path=Content,
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type
t:RadListBoxItem}}}"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadComboBox>
</Grid>
</Border>
</DataTemplate>

<!-- Pin_Parameter Pointer Template -->
<DataTemplate x:Key="PinParameterPointerTemplate">
<Border BorderBrush="DimGray" BorderThickness="0">
<Grid Margin="2">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="2*"  />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="0.5*"/>
</Grid.ColumnDefinitions>
<TextBlock Text="{Binding Path=ParamType}" FontSize="14" Grid.Column="0"/>
<TextBlock Text="{Binding Path=ParamName}" FontSize="14" Grid.Column="1"/>

<Grid Grid.Column="2" Width="Auto">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
```

```
<ColumnDefinition Width="2*"/>
</Grid.ColumnDefinitions>

<t:RadComboBox Width="Auto" Grid.Column="0" x:Name="ParamColumnBox"
SelectedValue="{Binding Path=TempParamMappingValue, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="PinName"
SelectedValuePath="PinName"
ItemsSource="{Binding Path=DataContext.Pins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged, RelativeSource={RelativeSource Mode=FindAncestor,
AncestorType={x:Type t:RadListBox}}}"
ClearSelectionButtonVisibility="Visible"
ClearSelectionButtonContent="No Pin">
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="MouseLeftButtonUp" Command="{Binding
Path=DataContext.TempParamValueChanged, RelativeSource={RelativeSource
Mode=FindAncestor, AncestorType={x:Type t:RadListBox}}}"
RaiseOnHandledEvents="True" CommandParameter="{Binding Path=Content,
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type
t:RadListBoxItem}}}"/>
</t:EventToCommandBehavior.EventBindings>
<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Setter Property="IsEnabled" Value="{Binding Path=IsMapped}"/>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>

<TextBox Text="{Binding Path=Value, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
FontSize="14" Width="Auto" Grid.Column="1"  >
<TextBox.Style>
<Style TargetType="TextBox">
<Setter Property="IsEnabled" Value="{Binding Path=IsMapped}"/>
<Setter Property="IsReadOnly" Value="True"/>
</Style>
</TextBox.Style>
</TextBox>

</Grid>

<t:RadComboBox Width="Auto" Grid.Column="3" x:Name="UnitColumnBox"
IsEnabled="{Binding Path=ParamMapping.IsUnitable, Mode=OneWay,
```

```
UpdateSourceTrigger=PropertyChanged}"
SelectedValue="{Binding Path=ParamUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=ParamMapping.Units, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged }">
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="MouseLeftButtonUp" Command="{Binding
Path=DataContext.ReadTempParamValue, RelativeSource={RelativeSource Mode=FindAncestor,
AncestorType={x:Type t:RadListBox}}}"
RaiseOnHandledEvents="True" CommandParameter="{Binding Path=Content,
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type
t:RadListBoxItem}}}"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadComboBox>
</Grid>
</Border>
</DataTemplate>

<!-- Manual Pointer Template -->
<DataTemplate x:Key="ManualPointerTemplate">
<Border BorderBrush="DimGray" BorderThickness="0">
<Grid Margin="2">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="2*"  />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="0.5*"/>
</Grid.ColumnDefinitions>
<TextBlock Text="{Binding Path=ParamType}" FontSize="14" Grid.Column="0"/>
<TextBlock Text="{Binding Path=ParamName}" FontSize="14" Grid.Column="1"/>

<TextBox Text="{Binding Path=Value, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
FontSize="14" Width="Auto" Grid.Column="2"  >
<TextBox.Style>
<Style TargetType="TextBox">
<Setter Property="Visibility" Value="Visible"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=IsEnum}" Value="true">
<Setter Property="Visibility" Value="Collapsed"/>
</DataTrigger>
</Style.Triggers>
</Style>
```

```
</TextBox.Style>
</TextBox>

<t:RadComboBox Width="Auto" Grid.Column="2" x:Name="ParamColumnBox"
SelectedItem="{Binding Path=Value, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=ParamOptions}"
ClearSelectionButtonVisibility="Visible"
ClearSelectionButtonContent="No Enum">
<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Setter Property="Visibility" Value="Collapsed"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=IsEnum}" Value="true">
<Setter Property="Visibility" Value="Visible"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>

<t:RadComboBox Width="Auto" Grid.Column="3" x:Name="UnitColumnBox"
SelectedValue="{Binding Path=ParamUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=ParamMapping.Units, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged }"
ClearSelectionButtonVisibility="Visible"
ClearSelectionButtonContent="No Unit">
</t:RadComboBox>
</Grid>
</Border>
</DataTemplate>



<!-- Mapping Pointer Template -->
<DataTemplate x:Key="MappingPointerTemplate">
<Border BorderBrush="DimGray" BorderThickness="0" >
<Grid Margin="2">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="2*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
```

```
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="0.5*"/>
</Grid.ColumnDefinitions>
<TextBlock Text="{Binding Path=ParamType}" FontSize="14" Grid.Column="0"/>
<TextBlock Text="{Binding Path=ParamName}" FontSize="14" Grid.Column="1"/>

<t:RadComboBox Width="Auto" Grid.Column="2" x:Name="PointerColumnBox"
SelectedValue="{Binding Path=Pointer, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Source={local:EnumBindingSource {x:Type
PointerEnumLocal:MappingPointer}}}">

</t:RadComboBox>
<t:RadComboBox  x:Name="ParamColumnBox"  Width="Auto" Grid.Column="3"
ClearSelectionButtonVisibility="Visible"
ClearSelectionButtonContent="No Mapping">

<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Setter Property="IsEnabled" Value="True"/>
<Setter Property="DisplayMemberPath" Value="ColumnName"/>
<Setter Property="ItemsSource" Value="{Binding Path=DataContext.TestParamters,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged, RelativeSource={RelativeSource
Mode=FindAncestor, AncestorType={x:Type t:RadListBox}}}"/>
<Setter Property="SelectedItem" Value="{Binding Path=ParamMapping, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>

<Style.Triggers>
<DataTrigger Binding="{Binding Path=Pointer}" Value="0">
<Setter Property="IsEnabled" Value="False"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>

<t:RadComboBox Width="Auto" Grid.Column="4" x:Name="UnitColumnBox"
IsEnabled="{Binding Path=ParamMapping.IsUnitable, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedValue="{Binding Path=ParamUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=ParamMapping.Units, Mode=OneWay,
```

```
UpdateSourceTrigger=PropertyChanged }">

</t:RadComboBox>
</Grid>
</Border>
</DataTemplate>

</ResourceDictionary>
```

CustomRadPane.xaml

```
<UserControl x:Class="Telerik.Windows.Controls.CustomRadPane"
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:MerlinTestStudio_Demo_Telerik.UserControls"
mc:Ignorable="d"
d:DesignHeight="300" d:DesignWidth="300">
<Grid>

</Grid>
</UserControl>
```

DllFileInfo.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.DllFileInfo"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="Auto"/>
```

```
</Grid.RowDefinitions>

<!--Dll File Info & Types List Box-->
<Grid Grid.Column="0" Grid.RowSpan="2">
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<!--Dll File Info-->
<t:GroupBox Header="Reference Info"  Grid.Row="1" Padding="5">
<Grid >
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>

<TextBlock Grid.Row="0"
Grid.Column="0"
Text="Version: "
FontWeight="Bold" />
<TextBlock Grid.Row="0"
Grid.Column="1"
Text="{Binding Path=DllFileDataObj.Version, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />

<TextBlock Grid.Row="1"
Grid.Column="0"
Text="Culture: "
FontWeight="Bold" />
<TextBlock Grid.Row="1"
Grid.Column="1"
Text="{Binding Path=DllFileDataObj.Culture, Mode=TwoWay,
```

```
UpdateSourceTrigger=PropertyChanged}" />

<TextBlock Grid.Row="2"
Grid.Column="0"
Text="Public Key Token: "
FontWeight="Bold" />
<TextBlock Grid.Row="2"
Grid.Column="1"
Text="{Binding Path=DllFileDataObj.PublicKeyToken, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />

<TextBlock Grid.Row="3"
Grid.Column="0"
Text="File Name: "
FontWeight="Bold" />
<TextBlock Grid.Row="3"
Grid.Column="1"
Text="{Binding Path=DllFileDataObj.FileName, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />

<TextBlock Grid.Row="4"
Grid.Column="0"
Text="Last Modified: "
FontWeight="Bold" />
<TextBlock Grid.Row="4"
Grid.Column="1"
Text="{Binding Path=DllFileDataObj.LastModified, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />

<TextBlock Grid.Row="5"
Grid.Column="0"
Text="File Type: "
FontWeight="Bold" />
<TextBlock Grid.Row="5"
Grid.Column="1"
Text="{Binding Path=DllFileDataObj.FileType, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />

<TextBlock Grid.Row="6"
Grid.Column="0"
Text="Description: "
FontWeight="Bold" />
```

```
<TextBlock Grid.Row="6"
Grid.Column="1"
Text="{Binding Path=DllFileDataObj.Description, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />
</Grid>
</t:GroupBox>

<!--Types List Box-->
<t:GroupBox Header="Types" Grid.Row="0">
<t:RadListBox x:Name="TypesListBox" ItemsSource="{Binding Path=DllFileDataObj.Types,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}" Background="Transparent"
SelectedItem="{Binding Path=SelectedType, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">
<t:RadListBox.ItemContainerStyle>
<Style TargetType="t:RadListBoxItem" >
<Setter Property="BorderBrush" Value="Transparent"/>
<Setter Property="BorderThickness" Value="1"/>
</Style>
</t:RadListBox.ItemContainerStyle>
<t:RadListBox.ItemTemplate>
<DataTemplate>
<Grid>
<t:Label Grid.Column="1" Content="{Binding Path=SelfType.Name, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="12"/>
</Grid>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>
</t:GroupBox>
</Grid>

<Grid Grid.Column="1" Grid.RowSpan="2">
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Selected Type Methods List Box-->
<t:GroupBox Header="Methods"  Grid.Row="0">
<t:RadListBox x:Name="MethodsListBox"
ItemsSource="{Binding Path=SelectedType.Methods, Mode=OneWay,
```

```
UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=SelectedMethod, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Background="Transparent">
<t:RadListBox.ItemContainerStyle>
<Style TargetType="t:RadListBoxItem" >
<Setter Property="BorderBrush" Value="Transparent"/>
<Setter Property="BorderThickness" Value="1"/>
</Style>
</t:RadListBox.ItemContainerStyle>
<t:RadListBox.ItemTemplate>
<DataTemplate>
<Grid>
<t:Label Grid.Column="1" Content="{Binding Path=Name, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="12"/>
</Grid>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>
</t:GroupBox>

<t:GroupBox Header="Method Parameters"  Grid.Row="1">
<t:RadListBox x:Name="ParametersListBox"
ItemsSource="{Binding  Path=SelectedMethodParameters, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
Background="Transparent">
<t:RadListBox.ItemContainerStyle>
<Style TargetType="t:RadListBoxItem" >
<Setter Property="BorderBrush" Value="Transparent"/>
<Setter Property="BorderThickness" Value="1"/>
</Style>
</t:RadListBox.ItemContainerStyle>
<t:RadListBox.ItemTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="2*"/>
</Grid.ColumnDefinitions>

<t:Label Grid.Column="1" Content="{Binding Path=ParameterType, Mode=OneWay,
```

```
UpdateSourceTrigger=PropertyChanged}"
HorizontalAlignment="Left" VerticalAlignment="Center" FontSize="12"/>

<t:Label Grid.Column="0" Content="{Binding Path=Name, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
HorizontalAlignment="Left" VerticalAlignment="Center" FontSize="12"/>
</Grid>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>
</t:GroupBox>
</Grid>

<t:GroupBox Header="Method Properties" Grid.Column="2" Grid.RowSpan="2">
<t:RadPropertyGrid Item="{Binding Path=SelectedMethod, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
</t:GroupBox>

</Grid>
</UserControl>
```

LicenceManager.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.LicenceManager"
x:Name="LicenceManagerUserControl"
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:MerlinTestStudio_Demo_Telerik.ViewModels"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<!--<local:MainWindowViewModel x:Key="MainWindowVM"/>-->

</UserControl.Resources>

<Grid>
<StackPanel Grid.Row="0" Grid.Column="2" Margin="5">
<!--<TextBlock Text="Manually Input Licence Key Here:" HorizontalAlignment="Center" />
<TextBox x:Name="InputKeytextbox" Width="Auto" Height="20" Margin="0 5 0 5" Text="{Binding
Source={StaticResource MainWindowVM}, Path=InputLicenceKeyText, Mode=TwoWay,
```

```
UpdateSourceTrigger=PropertyChanged}"/>
<Button x:Name="CheckKeyBtn" Content="Check Licence Key" Command="{Binding
Source={StaticResource MainWindowVM}, Path=CheckLicenceKey}" Margin="5" Width="120"
Height="20"/>
<TextBlock Text="License Key Validation Result:" Margin="0 20 0 0"/>
<TextBox x:Name="OutputResulttextbox" Width="Auto" Height="40" Text="{Binding
Source={StaticResource MainWindowVM}, Path=LicenceKeyValidationText, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" IsReadOnly="True"/>
<TextBlock Text="License Seed:" Margin="0 10 0 0"/>
<TextBox x:Name="SeedTextBox" Width="Auto" Height="40" Text="{Binding
Source={StaticResource MainWindowVM}, Path=LicenceKeySeed, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" IsReadOnly="True"/>-->


</StackPanel>
</Grid>
</UserControl>
```

PlaceHolderTextBox.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.PlaceHolderTextBox"
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:MerlinTestStudio_Demo_Telerik.UserControls"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
<Grid>
<TextBlock Text="{Binding Path=PlaceHolderText, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" VerticalAlignment="Center"
HorizontalAlignment="Center" FontSize="14" Foreground="White"/>
</Grid>
</UserControl>
```

Splash.xaml

```
<Window x:Class="MT.TestStudio.GUI.Splash"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
Title="Splash" Height="263" Width="523" WindowStartupLocation="CenterScreen">
<!--Height="5" Width="5"-->
<Window.Style>
<Style TargetType="{x:Type Window}">
```

```
<Setter Property="Topmost" Value="True"/>
<Setter Property="ShowInTaskbar" Value="False"/>
<Setter Property="WindowStyle" Value="None" />
<Setter Property="ResizeMode" Value="NoResize"/>
<Setter Property="AllowsTransparency" Value="True"/>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Window}">
<Border ClipToBounds="True">
<Grid ClipToBounds="True">
<AdornerDecorator>
<ContentPresenter Content="{TemplateBinding Content}"/>
</AdornerDecorator>
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</Window.Style>
<Grid Background="#2D2D30">
<!--<Grid.Background>
<ImageBrush ImageSource="Images/Logo White Background.png" />
</Grid.Background>-->
<DockPanel>
<Label DockPanel.Dock="Top" Margin="0,70,0,0" Content="Merlin Test Studio"
Foreground="White" FontSize="50" FontWeight="Bold" HorizontalContentAlignment="Center"/>
<StackPanel Orientation="Horizontal" DockPanel.Dock="Top" Margin="0,30,0,0">

<Label Visibility="Collapsed" Name="statuslbl" Content="" FontSize="22"  Foreground="White"
Margin="50,0,0,0" Height="50" VerticalAlignment="Bottom" HorizontalAlignment="Left"
Width="210" HorizontalContentAlignment="Center" />
<Label Name="versionlbl"  Content="{Binding Version, Mode=OneWay}"  FontSize="18"
FontWeight="Bold" Foreground="White" Margin="250,0,0,0" Height="50"
VerticalAlignment="Bottom" HorizontalAlignment="Right" Width="210"
HorizontalContentAlignment="Right"/>

</StackPanel>
<Rectangle Margin="0,24,0,0" Height="10" Stretch="Fill" Fill="#4C4C4F"
DockPanel.Dock="Bottom" />
</DockPanel>
</Grid>
```

```
</Window>
```

UserPreferences.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.UserPreferences"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<StackPanel  Grid.Row="0" Margin="5" >
<TextBlock Text="Developer Mode:" Padding="0 5 5 5" FontSize="14"/>
<t:RadToggleSwitchButton Margin="5"
HorizontalAlignment="Left" VerticalAlignment="Center"
UncheckedContent="User" CheckedContent="DevMode"
IsChecked="{Binding Path=IsDevModeActive, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
</StackPanel>

<StackPanel  Grid.Row="1" Margin="5" >
<TextBlock Text="Theme:" Padding="0 5 5 5" FontSize="14"/>
<t:RadComboBox MinWidth="100"
SelectedItem="{Binding Path=Theme, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=ThemeOptions, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" >
</t:RadComboBox>
</StackPanel>


<t:RadButton x:Name="SavePreferencesButton" Grid.Row="2"
Content="Apply" VerticalAlignment="Bottom"
Margin="5" FontSize="14" Padding="5"
Click="SavePreferencesButton_Click">
```

```
</t:RadButton>
</Grid>
</UserControl>
```

WindowStyled.xaml

```
<Window x:Class="MT.TestStudio.GUI.WindowStyled"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
mc:Ignorable="d"
TextElement.FontWeight="Regular"
TextElement.FontSize="10"
TextOptions.TextFormattingMode="Ideal"
TextOptions.TextRenderingMode="Auto"
Title="New Project" Height="768" Width="1024" WindowState="Maximized"
WindowStartupLocation="Manual" ResizeMode="CanResizeWithGrip">
<Window.Resources>



</Window.Resources>
<Grid>

</Grid>
</Window>
```

AttenuatorResultsViewer.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.AttenuatorResultsViewer"
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"
mc:Ignorable="d"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<t:BooleanToVisibilityConverter x:Key="FineGrainControlsVisibilityConverter"/>
<t:InvertedBooleanToVisibilityConverter x:Key="CourseGrainControlsVisibilityConverter" />
</UserControl.Resources>
```

```
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="5"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="3.5*"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<!--Data Selector-->
<Grid Grid.Row="0" Grid.ColumnSpan="3">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<TextBlock Text="Frequency:"  Grid.Column="0" Grid.Row="0" Foreground="White" Margin="5 0
0 0" FontSize="14"/>
<t:RadComboBox x:Name="FrequenciesList" HorizontalAlignment="Left" VerticalAlignment="Top"
MinWidth="100"
Grid.Column="0" Grid.Row="1" Margin="5" FontSize="14"
SelectedValue="{Binding Path=SelectedFrequency, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=AvailableFrequencies, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
ChangeChartDataCommand}" CommandParameter="Frequency"
RaiseOnHandledEvents="True"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadComboBox>
```

```
<TextBlock Text="Amp Configuration:"  Grid.Column="1" Grid.Row="0" Foreground="White"
Margin="5 0 0 0" FontSize="14"/>
<t:RadComboBox x:Name="AmpConfigsList" Height="Auto" MinWidth="150"
HorizontalAlignment="Left" VerticalAlignment="Top"
Grid.Column="1" Grid.Row="1" Margin="5" FontSize="14"
SelectedValue="{Binding Path=SelectedAmpConfig, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=AvailableAmpConfigs, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
ChangeChartDataCommand}" CommandParameter="AmpConfig"
RaiseOnHandledEvents="True"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadComboBox>

<TextBlock Text="Group:"  Grid.Column="2" Grid.Row="0" Foreground="White" Margin="5 0 0 0"
FontSize="14"/>
<t:RadComboBox  x:Name="GroupsComboBox" Margin="5" FontSize="14"
Grid.Row="1" Grid.Column="2" HorizontalAlignment="Left" VerticalAlignment="Top"
Height="Auto" Width="100"
SelectedValue="{Binding Path=SelectedGroup, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=AvailableGroups, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
ChangeChartDataCommand}" CommandParameter="Group" RaiseOnHandledEvents="True"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadComboBox>

<TextBlock Text="Level:"  Grid.Column="4" Grid.Row="0" Foreground="White" Margin="5 0 0 0"
FontSize="14"/>
<t:RadComboBox  x:Name="LevelsComboBox" Margin="5" FontSize="14"
Grid.Row="1" Grid.Column="4" HorizontalAlignment="Left" VerticalAlignment="Top"
Height="Auto" Width="100"
SelectedValue="{Binding Path=SelectedLevel, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=AvailableLevels, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
<t:EventToCommandBehavior.EventBindings>
```

```
<t:EventBinding EventName="SelectionChanged" Command="{Binding
ChangeChartDataCommand}" CommandParameter="Level" RaiseOnHandledEvents="True"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadComboBox>

<TextBlock Text="Waveform:"  Grid.Column="5" Grid.Row="0" Foreground="White" Margin="5 0 0
0" FontSize="14"/>
<t:RadComboBox  x:Name="WaveformsComboBox" Margin="5" FontSize="14"
Grid.Row="1" Grid.Column="5" HorizontalAlignment="Left" VerticalAlignment="Top"
Height="Auto" Width="100"
SelectedValue="{Binding Path=SelectedWaveform, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=AvailableWaveforms, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
ChangeChartDataCommand}" CommandParameter="Waveform"
RaiseOnHandledEvents="True"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadComboBox>

<t:RadButton x:Name="exportDataBtn" Grid.Row="1" Grid.Column="5" Margin="5"
VerticalAlignment="Center" HorizontalAlignment="Right"
ToolTip="Exports the active chart data." FontSize="14"
IsEnabled="{Binding Path=IsChartDataLoaded, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
Click="RadButton_Click">
<t:RadButton.Content>
<WrapPanel>
<t:RadGlyph Glyph="&#xe109;" Foreground="White" Margin="0 0 5 0"/>
<TextBlock Text="Export"/>
</WrapPanel>
</t:RadButton.Content>
</t:RadButton>
</Grid>

<!--Chart Data Viewer-->
<t:RadCartesianChart x:Name="Chart" Grid.Row="1"  Grid.Column="0"
Background="Transparent" Foreground="White" Palette="Windows8"
SelectionPalette="Windows8Selected">

<t:RadCartesianChart.Resources>
```

```
<DataTemplate x:Key="trackBallTemplate">
<Rectangle Width="10" Height="10" Fill="Orange" Stroke="Black" StrokeThickness="1">
<Rectangle.LayoutTransform>
<RotateTransform Angle="45" />
</Rectangle.LayoutTransform>
</Rectangle>
</DataTemplate>

</t:RadCartesianChart.Resources>
<t:RadCartesianChart.Behaviors>
<t:ChartTrackBallBehavior x:Name="trackballBehavior"
SnapMode="ClosestPoint"
ShowIntersectionPoints="True"
SnapSinglePointPerSeries="True"
ShowTrackInfo="True"
>
</t:ChartTrackBallBehavior>
<t:ChartPanAndZoomBehavior x:Name="chartPanAndZoomBehavior" ZoomMode="Both" />
</t:RadCartesianChart.Behaviors>
<t:RadCartesianChart.VerticalAxis >
<t:LinearAxis Title="Attenuation (dB)" HorizontalLocation="Left" ElementBrush="DodgerBlue">
</t:LinearAxis>
</t:RadCartesianChart.VerticalAxis>
<t:RadCartesianChart.HorizontalAxis>
<t:LinearAxis Foreground="White" Title="{Binding Path=HorizontalAxisTitle, Mode=OneWay}">
<t:LinearAxis.LabelStyle>
<Style TargetType="TextBlock">
<Setter Property="Foreground" Value="White"/>
</Style>
</t:LinearAxis.LabelStyle>
</t:LinearAxis>
</t:RadCartesianChart.HorizontalAxis>
<t:RadCartesianChart.Series>
<t:ScatterLineSeries x:Name="KeyValueSeries" Stroke="DodgerBlue" StrokeThickness="2"
XValueBinding="Key" YValueBinding="Value"
ItemsSource="{Binding Path= ChartData, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
TrackBallTemplate="{StaticResource trackBallTemplate}">
<t:ScatterLineSeries.TrackBallInfoTemplate>
<DataTemplate>
<StackPanel Background="White" Margin="3" Width="Auto">
<StackPanel Orientation="Horizontal">
```

```
<TextBlock Text="X=" FontWeight="Bold" Foreground="DodgerBlue"/>
<TextBlock Text="{Binding DataPoint.XValue}" />
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="Y=" FontWeight="Bold" Foreground="DodgerBlue"/>
<TextBlock Text="{Binding DataPoint.YValue}" />
</StackPanel>
</StackPanel>
</DataTemplate>
</t:ScatterLineSeries.TrackBallInfoTemplate>
<t:ScatterLineSeries.Style>
<Style TargetType="t:ScatterLineSeries">
<Setter Property="Visibility" Value="Collapsed" />
<Style.Triggers>
<DataTrigger Binding="{Binding ElementName=Y_Delta_Switch, Path=IsChecked}"
Value="False">
<Setter Property="Visibility" Value="Visible" />
</DataTrigger>
</Style.Triggers>
</Style>
</t:ScatterLineSeries.Style>
</t:ScatterLineSeries>
<t:ScatterLineSeries x:Name="RegressedXSeries" Stroke="LightBlue" StrokeThickness="2"
XValueBinding="Regressed" YValueBinding="Value"
ItemsSource="{Binding Path= ChartData, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
TrackBallTemplate="{StaticResource trackBallTemplate}" >
<t:ScatterLineSeries.TrackBallInfoTemplate>
<DataTemplate>
<StackPanel Background="White" Margin="3" Width="Auto">
<StackPanel Orientation="Horizontal">
<TextBlock Text="X=" FontWeight="Bold" Foreground="LightBlue" />
<TextBlock Text="{Binding DataPoint.XValue}" />
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="Y=" FontWeight="Bold" Foreground="LightBlue" />
<TextBlock Text="{Binding DataPoint.YValue}" />
</StackPanel>
</StackPanel>
</DataTemplate>
</t:ScatterLineSeries.TrackBallInfoTemplate>
<t:ScatterLineSeries.Style>
```

```
<Style TargetType="t:ScatterLineSeries">
<Setter Property="Visibility" Value="Collapsed" />
<Style.Triggers>
<DataTrigger Binding="{Binding ElementName=X_Regressed_Switch, Path=IsChecked}"
Value="True">
<Setter Property="Visibility" Value="Visible" />
</DataTrigger>
<DataTrigger Binding="{Binding ElementName=Y_Delta_Switch, Path=IsChecked}"
Value="True">
<Setter Property="Visibility" Value="Collapsed" />
</DataTrigger>
</Style.Triggers>
</Style>
</t:ScatterLineSeries.Style>
</t:ScatterLineSeries>
<t:ScatterLineSeries x:Name="DeltaSeries" Stroke="Orange" StrokeThickness="2"
XValueBinding="Key" YValueBinding="Delta"
ItemsSource="{Binding Path= DeltaChartData, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
TrackBallTemplate="{StaticResource trackBallTemplate}" >
<t:ScatterLineSeries.TrackBallInfoTemplate>
<DataTemplate>
<StackPanel Background="White" Margin="3" Width="Auto">
<StackPanel Orientation="Horizontal">
<TextBlock Text="X=" FontWeight="Bold" Foreground="Orange" />
<TextBlock Text="{Binding DataPoint.XValue, StringFormat={}{0:F3} }" />
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="Y=" FontWeight="Bold" Foreground="Orange" />
<TextBlock Text="{Binding DataPoint.YValue}" />
</StackPanel>
</StackPanel>
</DataTemplate>
</t:ScatterLineSeries.TrackBallInfoTemplate>
<t:ScatterLineSeries.VerticalAxis>
<t:LinearAxis Title="Delta" HorizontalLocation="Right" ElementBrush="Orange">
<t:LinearAxis.Style>
<Style TargetType="t:LinearAxis">
<Setter Property="Visibility" Value="Collapsed" />
<Style.Triggers>
<DataTrigger Binding="{Binding ElementName=Y_Delta_Switch, Path=IsChecked}"
Value="True">
```

```
<Setter Property="Visibility" Value="Visible" />
</DataTrigger>
</Style.Triggers>
</Style>
</t:LinearAxis.Style>
</t:LinearAxis>
</t:ScatterLineSeries.VerticalAxis>
<t:ScatterLineSeries.Style>
<Style TargetType="t:ScatterLineSeries">
<Setter Property="Visibility" Value="Collapsed" />
<Style.Triggers>
<DataTrigger Binding="{Binding ElementName=Y_Delta_Switch, Path=IsChecked}"
Value="True">
<Setter Property="Visibility" Value="Visible" />
</DataTrigger>
</Style.Triggers>
</Style>
</t:ScatterLineSeries.Style>
</t:ScatterLineSeries>
</t:RadCartesianChart.Series>
<t:RadCartesianChart.Grid>
<t:CartesianChartGrid MajorXLineDashArray="2 5"    MajorYLineDashArray="2 5"
MajorLinesVisibility="XY" />
</t:RadCartesianChart.Grid>

</t:RadCartesianChart>

<!--Vertical Grid Splitter-->
<GridSplitter Width="5" Grid.Row="1" Grid.Column="1" HorizontalAlignment="Stretch"
Background="Transparent" />

<!--Point Data View-->
<t:RadGridView x:Name="ChartDataGridView" Grid.Row="1" Grid.Column="2"
ItemsSource="{Binding Path=ChartData, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
SelectionChanged="RadGridView_SelectionChanged"
Height="Auto" Width="Auto" MinWidth="100"
AutoGenerateColumns ="False"
EditTriggers="None"
EnableColumnVirtualization="True"
EnableRowVirtualization="True"
RowIndicatorVisibility="Collapsed"
```

```
ShowGroupPanel="False"
CanUserSortColumns="False"
IsFilteringAllowed="False"
GroupRenderMode="Flat"
FontSize="14" RowHeight="25"
DataLoadMode="Asynchronous">
<t:RadGridView.Columns>
<t:GridViewDataColumn Header="X (dB)" DataMemberBinding="{Binding Key}"
Style="{StaticResource RegressionDataColumnStyle}" IsReadOnly="True" DataFormatString="{}
{0:N3}" MinWidth="60"/>
<t:GridViewDataColumn Header="X (V)" DataMemberBinding="{Binding Key}"
Style="{StaticResource CourseDataColumnStyle}" IsReadOnly="True" DataFormatString="{}
{0:N3}" MinWidth="60"/>
<t:GridViewDataColumn Header="Y (dB)" DataMemberBinding="{Binding Value}"
IsReadOnly="True" DataFormatString="{} {0:N3}" MinWidth="60"/>
<t:GridViewDataColumn Header="rX (Calculated)" Style="{StaticResource
RegressionDataColumnStyle}"
DataMemberBinding="{Binding Regressed, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
IsReadOnly="True" DataFormatString="{} {0:N3}" MinWidth="60"/>
<t:GridViewDataColumn Header="Delta" DataMemberBinding="{Binding Delta}"
IsReadOnly="True" DataFormatString="{} {0:N3}" MinWidth="60"/>
<t:GridViewCheckBoxColumn Header="PassFlag" DataMemberBinding="{Binding PassFlag,
Mode=OneWay}" IsReadOnly="True" >
<t:GridViewCheckBoxColumn.Style>
<Style TargetType="t:GridViewCheckBoxColumn" >
<Setter Property="IsVisible" Value="True"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=ShowPolynomialRegressionControls, Mode=OneWay}"
Value="True">
<Setter Property="IsVisible" Value="False"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:GridViewCheckBoxColumn.Style>
</t:GridViewCheckBoxColumn>
</t:RadGridView.Columns>
</t:RadGridView>

<!--Below of chart area-->
<Grid Grid.Row="2" Grid.ColumnSpan="3"  >
<Grid.ColumnDefinitions>
```

```
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<t:RadToggleSwitchButton x:Name="Y_Delta_Switch" Margin="5"  Grid.Column="0"
HorizontalAlignment="Left" VerticalAlignment="Center"
UncheckedContent="dB" CheckedContent="Delta"
IsEnabled="{Binding Path=IsChartDataLoaded, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"/>

<WrapPanel Grid.Column="1" Margin="5" HorizontalAlignment="Left" VerticalAlignment="Center"
>
<WrapPanel.Style>
<Style TargetType="WrapPanel">
<Setter Property="Visibility" Value="Collapsed"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=ShowPolynomialRegressionControls, Mode=OneWay}"
Value="True">
<Setter Property="Visibility" Value="Visible"/>
</DataTrigger>
</Style.Triggers>
</Style>
</WrapPanel.Style>

<t:RadToggleSwitchButton x:Name="X_Regressed_Switch"  Margin="0 0 5 0"
CheckedContent="rX" UncheckedContent="X"
IsEnabled="{Binding Path=IsChartDataLoaded, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
</t:RadToggleSwitchButton>
<TextBox x:Name="RegressionEquation" FontSize="14" IsReadOnly="True"
Text="{Binding Path=RegressionFormula, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"/>
</WrapPanel>
</Grid>

</Grid>
</UserControl>
```

CalDataInfo.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.CalDataInfo"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
<UserControl.Resources>

</UserControl.Resources>

<Grid >
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<t:GroupBox Header="CalDataV5 Info" Grid.Column="0" Grid.Row="0" Padding="5">
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<TextBlock Grid.Column="0" Grid.Row="0" Text="Product: " Style="{StaticResource
ProductInfoLabelStyle}"/>
<TextBlock Grid.Column="0" Grid.Row="1" Text="Revision: " Style="{StaticResource
ProductInfoLabelStyle}"/>
<TextBlock Grid.Column="0" Grid.Row="2" Text="Test Program: " Style="{StaticResource
ProductInfoLabelStyle}"/>

<TextBlock Grid.Column="1" Grid.Row="4" Text="Noise Calibration Sample Rate: "
```

```
Style="{StaticResource ProductInfoLabelStyle}" TextWrapping="Wrap"
HorizontalAlignment="Left"/>

<TextBox Grid.Column="1" Grid.Row="0" Text="{Binding Product, Mode=OneWay}"
Style="{StaticResource ProductInfoInputStyle}" IsReadOnly="True"/>
<TextBox Grid.Column="1" Grid.Row="1" Text="{Binding Revision, Mode=OneWay}"
Style="{StaticResource ProductInfoInputStyle}" IsReadOnly="True"/>
<TextBox Grid.Column="1" Grid.Row="2" Text="{Binding TestProgram, Mode=OneWay}"
Style="{StaticResource ProductInfoInputStyle}" IsReadOnly="True"/>
<WrapPanel Grid.Column="1" Grid.Row="3">
<TextBlock Grid.Column="0" Grid.Row="3" Text="Calibration Data Version: "
Style="{StaticResource ProductInfoLabelStyle}" TextWrapping="Wrap"/>
<t:RadNumericUpDown Grid.Column="1" Grid.Row="3" Value="{Binding CalibrationDataVersion,
Mode=OneWay, UpdateSourceTrigger=LostFocus}"
Margin="5 5 5 5" HideTrailingZeros="True" HorizontalContentAlignment="Left"
VerticalAlignment="Center" HorizontalAlignment="Left" IsReadOnly="True"/>
</WrapPanel>

<CheckBox Grid.Column="0" Grid.Row="3" Content="Calibrate RF" IsChecked="{Binding
CalibrateRF, Mode=OneWay, UpdateSourceTrigger=LostFocus}"
Margin="5" VerticalAlignment="Center" HorizontalAlignment="Left" Foreground="White"
IsHitTestVisible="false"
Style="{StaticResource CheckBoxStyle}" />
<CheckBox Grid.Column="0" Grid.Row="4" Content="Calibrate Noise" IsChecked="{Binding
CalibrateNoise, Mode=OneWay, UpdateSourceTrigger=LostFocus}"
Margin="5" VerticalAlignment="Center" HorizontalAlignment="Left" Foreground="White"
IsHitTestVisible="false"
Style="{StaticResource CheckBoxStyle}"/>
<CheckBox Grid.Column="0" Grid.Row="5" Content="Calibrate VNA" IsChecked="{Binding
CalibrateVNA, Mode=OneWay, UpdateSourceTrigger=LostFocus}"
Margin="5" VerticalAlignment="Center" HorizontalAlignment="Left" Foreground="White"
IsHitTestVisible="false"
Style="{StaticResource CheckBoxStyle}"/>

<t:RadNumericUpDown Grid.Column="1" Grid.Row="5" Value="{Binding
NoiseCalibrationSampleRate, Mode=OneWay, UpdateSourceTrigger=LostFocus}" Margin="5 0 5
5" HideTrailingZeros="True" HorizontalContentAlignment="Left" VerticalAlignment="Center"
IsReadOnly="True"  />
</Grid>
</t:GroupBox>
```

```
<t:GroupBox Header="Cal Points V5" Grid.Column="1" Grid.RowSpan="3">
<t:RadGridView x:Name="CalPointsGridView"
ItemsSource="{Binding Path=CalPoints, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
Height="Auto" Width="Auto"
AutoGenerateColumns ="True"
AutoGeneratingColumn="CalPointsGridView_AutoGeneratingColumn"
EditTriggers="None"
RowHeight="30"
SelectionUnit="FullRow"
SelectionMode="Extended"
EnableColumnVirtualization="True"
EnableRowVirtualization="True"
RowIndicatorVisibility="Collapsed"
GroupRenderMode="Flat"
IsPropertyChangedAggregationEnabled="False"
FontSize="14">

</t:RadGridView>
</t:GroupBox>

<t:GroupBox Header="Total Entries / Number of Cal-Points" Grid.Column="0" Grid.Row="2"
Background="Transparent" HorizontalContentAlignment="Center" VerticalAlignment="Center">
<TextBlock VerticalAlignment="Center" HorizontalAlignment="Center" FontSize="20" Margin="10"
Foreground="White"
Text="{Binding Path=CalPoints.Count, StringFormat={}{0} Entries, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of calibration points in this Calibration Definition file.">
</TextBlock>
</t:GroupBox>

<t:GroupBox Header="Rf110 Amplifiers To Calibrate" Grid.Column="0" Grid.Row="1">
<t:RadListBox ItemsSource="{Binding Path=Rf110AmplifiersToCalibrate, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">

</t:RadListBox>
</t:GroupBox>

</Grid>
</UserControl>
```

CalibrationConfiguratorView.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.CalibrationConfiguratorView"
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:MerlinTestStudio_Demo_Telerik.UserControls"
xmlns:conv="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Converters"
xmlns:eSourceExt="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
xmlns:e="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Models"
xmlns:cd="clr-namespace:MerlinTest.Common.Types;assembly=MerlinTest.Common.Types"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800" Unloaded="UserControl_Unloaded" >

<UserControl.InputBindings>
<KeyBinding Key="Esc" Command="{Binding ClearCalPointSelectionCommand}"/>
</UserControl.InputBindings>

<UserControl.Resources>
<conv:FrequencyConverter x:Key="FreqConvUnit"/>
<conv:NullToColorConverter x:Key="NullToColorConverter" />
<t:BooleanToVisibilityConverter x:Key="VisToBool"/>
</UserControl.Resources>

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<t:RadExpander Header="CalDataV5 Properties" VerticalContentAlignment="Top"
ExpandDirection="Right" t:AnimationManager.IsAnimationEnabled="False"
IsExpanded="True" Grid.Column="0" Grid.Row="0" MaxWidth="360" >
<t:GroupBox Header="CalDataV5" FontSize="14" BorderBrush="#FF2D2D30"
BorderThickness="2"  >
<Grid>
```

```
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Cal Data Info Menu-->
<t:RadMenu Grid.Row="0">
<t:RadMenuItem Header="Compile" Command="{Binding Path=CompileCalDataV5Command}"
ToolTip="Compile a CalDataV5 file (.xml)" />
<t:RadMenuItem Header="Preview"
Visibility="{Binding Path=MVM.Preferences.IsDevModeActive, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource VisToBool}}"
Command="{Binding Path=UserCalPreviewCommand}"
ToolTip="View this configuration as it would generate in the UserCal using the current Pin Map.">
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadGlyph Glyph="&#xe401;" ToolTip="{Binding Path=CurrentCalConfig.InfoToolTip,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}" />

</t:RadMenu>

<!--Cal Data Product Info-->
<t:GroupBox Header="Product Info"  Grid.Row="1" >
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<CheckBox Grid.Column="0" Grid.Row="3" Content="Calibrate RF" IsChecked="{Binding
Path=CurrentCalConfig.CalibrateRF, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
Margin="5" VerticalAlignment="Center" HorizontalAlignment="Left" Foreground="White"
Style="{StaticResource CheckBoxStyle}" />
```

```
<CheckBox Grid.Column="0" Grid.Row="4" Content="Calibrate Noise" IsChecked="{Binding
Path=CurrentCalConfig.CalibrateNoise, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Margin="5" VerticalAlignment="Center" HorizontalAlignment="Left" Foreground="White"
Style="{StaticResource CheckBoxStyle}"/>
<CheckBox Grid.Column="0" Grid.Row="5" Content="Calibrate VNA" IsChecked="{Binding
Path=CurrentCalConfig.CalibrateVNA, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"

Margin="5" VerticalAlignment="Center" HorizontalAlignment="Left" Foreground="White"
Style="{StaticResource CheckBoxStyle}"/>

<WrapPanel Grid.Column="1" Grid.Row="4" >
<TextBlock Text="Sample Rate:" Style="{StaticResource ProductInfoLabelStyle}"
TextWrapping="Wrap" HorizontalAlignment="Left"/>
<t:RadNumericUpDown  HorizontalContentAlignment="Left" VerticalAlignment="Center"
Margin="5 0 5 5"
Value="{Binding Path=CurrentCalConfig.NoiseCalibrationSampleRate, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource FreqConvUnit},
ConverterParameter={} }"
HideTrailingZeros="True" />
</WrapPanel>

</Grid>
</t:GroupBox>

<!--Amps To Calibrate List-->
<t:GroupBox Header="Amplifiers To Calibrate" Grid.Row="2" FontSize="14">
<t:RadListBox Grid.Row="2" ItemsSource="{Binding Path=CurrentCalConfig.AmpConfigModelList,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}" Background="Transparent">
<t:RadListBox.ItemContainerStyle>
<Style TargetType="t:RadListBoxItem" >
<Setter Property="BorderBrush" Value="Transparent"/>
<Setter Property="BorderThickness" Value="1"/>
</Style>
</t:RadListBox.ItemContainerStyle>
<t:RadListBox.ItemTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
```

```
<t:Label Grid.Column="1" Content="{Binding Path=Name, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="12"/>
<CheckBox Grid.Column="0" IsChecked="{Binding Path=IsAdded, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
HorizontalAlignment="Left" VerticalAlignment="Center" Style="{StaticResource CheckBoxStyle}"/>
</Grid>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>
</t:GroupBox>
</Grid>
</t:GroupBox>
</t:RadExpander>

<!--CalPoints Editor-->
<t:GroupBox Header="Cal Points"  Grid.Column="1" Grid.Row="0" FontSize="14"
BorderBrush="#FF2D2D30" BorderThickness="2">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu Grid.Row="0" >
<t:RadMenuItem Header="Insert" Command="{Binding Path=InsertCalPointCommand}"
CommandParameter="{Binding ElementName=CalPointGridView, Path=SelectedItems}">
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe11f;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Header="Delete" Command="{Binding Path=RemoveCalPointsCommand}"
CommandParameter="{Binding ElementName=CalPointGridView, Path=SelectedItems}">
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe10c;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Duplicate" Command="{Binding Path=DuplicateCalPointCommand}"
IsEnabled="True" CommandParameter="{Binding ElementName=CalPointGridView,
Path=SelectedItems}">
<t:RadMenuItem.Icon >
```

```
<t:RadGlyph Glyph="&#xe65d;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadComboBox x:Name="GlobalUnitComboBox"  ToolTip="Select a unit for editing frequency."
SelectedValue="{Binding Path=CurrentCalConfig.GlobalUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=UnitList, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=CurrentCalConfig.CalPointModelCollection.Count,
StringFormat={}{0} Entries, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of calibration points in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>
<t:RadButton Content="Clear All" Grid.Row="0" HorizontalAlignment="Right"
ToolTip="Clears all of the CalPoints; Cannot Undo this action."
Command="{Binding ClearAllCommand}"/>

<t:RadGridView x:Name="CalPointGridView" Grid.Row="1"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Visible"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

ClipboardCopyMode="Cells"
ClipboardPasteMode="Cells, OverwriteWithEmptyValues"
PastingCellClipboardContent="CalPointGridView_PastingCellClipboardContent"
PreparingCellForEdit="CalPointGridView_PreparingCellForEdit"
CellEditEnded="CalPointGridView_CellEditEnded"
Deleting="CalPointGridView_Deleting"
Loaded="CalPointGridView_Loaded"

RowHeight="30"
AllowDrop="False"
```

```
CanUserDeleteRows="True"
SelectionUnit="Mixed"
SelectionMode="Extended"
SelectedItem="{Binding Path=SelectedCalPoint, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
eSourceExt:GridViewSelectionUtilities.SelectedItems="{Binding ManySelectedCalPoints}"
ItemsSource="{Binding Path=CurrentCalConfig.CalPointModelCollection, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
RowStyle="{StaticResource CalPointGridViewRowStyle}">

<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>

<t:RadGridView.Resources>
<DataTemplate x:Key="DraggedItemTemplate">
<StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="Dragging:" />
<TextBlock Text="{Binding CurrentDraggedItem}"
FontWeight="Bold" />
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="{Binding CurrentDropPosition}"
FontWeight="Bold"
MinWidth="45" />
<TextBlock Text=", ("
Foreground="Gray" />
<TextBlock Text="{Binding CurrentDraggedOverItem}" />
<TextBlock Text=")"
Foreground="Gray" />
</StackPanel>
</StackPanel>
</DataTemplate>
</t:RadGridView.Resources>

<t:RadGridView.Columns>
<t:GridViewComboBoxColumn Header="Cal Type" Width="Auto" HeaderTextWrapping="Wrap"
HeaderTextAlignment="Center"
```

```
DataMemberBinding="{Binding Path=CalibrationType, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Source={eSourceExt:EnumBindingSource {x:Type cd:CalType}}}"
EditTriggers="CellClick" EditorStyle="{StaticResource DropDownOnCellEdit}">
</t:GridViewComboBoxColumn>
<t:GridViewComboBoxColumn Header="Source" Width="60" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Source, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSourceBinding="{Binding Source={eSourceExt:EnumBindingSource {x:Type
e:SourceOptions}}}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
</t:GridViewComboBoxColumn>
<!--Src Pin Below-->
<t:GridViewComboBoxColumn Header="Src Pin" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=SrcPin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="GetPinLabelToUse"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.AvailablePins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
<t:GridViewComboBoxColumn.CellTemplate>
<DataTemplate>
<TextBlock Foreground="{Binding Path=SrcPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource NullToColorConverter}}" >
<TextBlock.Text>
<PriorityBinding>
<Binding Path="SrcPin.GetPinLabelToUse" Mode="OneWay"
UpdateSourceTrigger="PropertyChanged"  IsAsync="True" />
<Binding Path="SrcPinName" Mode="OneWay" UpdateSourceTrigger="PropertyChanged"
IsAsync="True" />
</PriorityBinding>
</TextBlock.Text>
</TextBlock>
</DataTemplate>
</t:GridViewComboBoxColumn.CellTemplate>
<t:GridViewComboBoxColumn.CellStyle>
<Style TargetType="t:GridViewCell">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=SrcPin, Mode=OneWay,
```

```
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="ToolTip" Value="Pin not found in project"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:GridViewComboBoxColumn.CellStyle>
</t:GridViewComboBoxColumn>
<!--Meas Pin Below-->
<t:GridViewComboBoxColumn Header="Meas Pin" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=MeasPin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="GetPinLabelToUse"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.AvailablePins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
<t:GridViewComboBoxColumn.CellTemplate>
<DataTemplate>
<TextBlock Foreground="{Binding Path=MeasPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource NullToColorConverter}}" >
<TextBlock.Text>
<PriorityBinding>
<Binding Path="MeasPin.GetPinLabelToUse" Mode="OneWay"
UpdateSourceTrigger="PropertyChanged"  IsAsync="True" />
<Binding Path="MeasPinName" Mode="OneWay" UpdateSourceTrigger="PropertyChanged"
IsAsync="True" />
</PriorityBinding>
</TextBlock.Text>
</TextBlock>
</DataTemplate>
</t:GridViewComboBoxColumn.CellTemplate>
<t:GridViewComboBoxColumn.CellStyle>
<Style TargetType="t:GridViewCell">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=MeasPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="ToolTip" Value="Pin not found in project"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:GridViewComboBoxColumn.CellStyle>
```

```
</t:GridViewComboBoxColumn>

<t:GridViewCheckBoxColumn Header="Use Hf Path" Width="50" HeaderTextWrapping="Wrap"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=UseHfPath, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
AutoSelectOnEdit="True" >
</t:GridViewCheckBoxColumn>
<t:GridViewDataColumn Header="Frequency" Width="100" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Frequency, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
CellStyle="{StaticResource WarningGridViewCellStyle}"
TextAlignment="Center" >
</t:GridViewDataColumn>
<t:GridViewDataColumn Header="Device Expected Input Level" Width="100"
HeaderTextWrapping="Wrap" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=DeviceExpectedInputLevel, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" TextAlignment="Center" >
</t:GridViewDataColumn>
<t:GridViewDataColumn Header="Device Expected Output Level" Width="100"
HeaderTextWrapping="Wrap" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=DeviceExpectedOutputLevel, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" TextAlignment="Center" >
</t:GridViewDataColumn>
<t:GridViewComboBoxColumn Header="Meas Filter" Width="70" HeaderTextWrapping="Wrap"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=MeasureFilter, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSourceBinding="{Binding Source={eSourceExt:EnumBindingSource {x:Type
e:MeasureFilterOptions}}}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
</t:GridViewComboBoxColumn>
<t:GridViewDataColumn Header="Meas Atten" Width="60" HeaderTextWrapping="Wrap"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=MeasureAttenuation, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewComboBoxColumn Header="Do Atten Cal" Width="Auto"
HeaderTextWrapping="Wrap"  HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=DoAttenCal, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
```

```
DisplayMemberPath="Key"
SelectedValueMemberPath="Value"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.DoAttenCalKvPs, Mode=OneTime}"
EditTriggers="CellClick">
</t:GridViewComboBoxColumn>
<t:GridViewComboBoxColumn Header="Src Cal Atten" Width="Auto"
HeaderTextWrapping="Wrap"  HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=SrcCalAttenSettings, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="Value"
SelectedValueMemberPath="Key"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.AttenuationSettingsKvPs,
Mode=OneTime}"
EditTriggers="CellClick">
</t:GridViewComboBoxColumn>
<t:GridViewComboBoxColumn Header="Rf110LfSrcAmp" Width="Auto"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Rf110LfSrcAmp, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="Name"
SelectedValueMemberPath="Config"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.CurrentCalConfig.AmpConfigModelList,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick" >
<t:GridViewComboBoxColumn.EditorStyle>
<Style TargetType="t:RadComboBox">
<Setter Property="ClearSelectionButtonVisibility" Value="Visible" />
<Setter Property="ClearSelectionButtonContent" Value="Clear" />
<Setter Property="ItemContainerStyle" >
<Setter.Value >
<Style TargetType="t:RadComboBoxItem">
<Setter Property="IsHitTestVisible" Value="{Binding IsAdded, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"/>
<Setter Property="Visibility" Value="Visible"/>
<Style.Triggers>
<DataTrigger Binding="{Binding IsAdded, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="False">
<Setter Property="Foreground" Value="IndianRed"/>
<Setter Property="Visibility" Value="Collapsed"/>
```

```
</DataTrigger>
</Style.Triggers>
</Style>
</Setter.Value>
</Setter>
</Style>
</t:GridViewComboBoxColumn.EditorStyle>
</t:GridViewComboBoxColumn>
<t:GridViewComboBoxColumn Header="Rf110LfMeasureAmp" Width="Auto"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Rf110LfMeasureAmp, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="Value" SelectedValueMemberPath="Key"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.LfMeasureAmpKvPs, Mode=OneTime}"
EditTriggers="CellClick" EditorStyle="{StaticResource NullableEnumRadComboBoxStyle}">
</t:GridViewComboBoxColumn>
<t:GridViewComboBoxColumn Header="Rf110HfSrcAmp" Width="Auto"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Rf110HfSrcAmp, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="Value" SelectedValueMemberPath="Key"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.HfAmpKvPs, Mode=OneTime}"
EditTriggers="CellClick" EditorStyle="{StaticResource NullableEnumRadComboBoxStyle}">
</t:GridViewComboBoxColumn>
<t:GridViewComboBoxColumn Header="Rf110HfMeasAmp" Width="Auto"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Rf110HfMeasAmp, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="Value" SelectedValueMemberPath="Key"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.HfAmpKvPs, Mode=OneTime}"
EditTriggers="CellClick" EditorStyle="{StaticResource NullableEnumRadComboBoxStyle}">
</t:GridViewComboBoxColumn>
<t:GridViewComboBoxColumn Header="Rf210LfAllAmp" Width="Auto"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Rf210LfAllAmp, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="Value" SelectedValueMemberPath="Key"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.LfAllAmpKvPs, Mode=OneTime}"
```

```
EditTriggers="CellClick" EditorStyle="{StaticResource NullableEnumRadComboBoxStyle}">
</t:GridViewComboBoxColumn>
<t:GridViewComboBoxColumn Header="Rf210HfAllAmp" Width="Auto"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Rf210HfAllAmp, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="Value" SelectedValueMemberPath="Key"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.HfAllAmpKvPs, Mode=OneTime}"
EditTriggers="CellClick" EditorStyle="{StaticResource NullableEnumRadComboBoxStyle}">
</t:GridViewComboBoxColumn>

<t:GridViewComboBoxColumn Header="Waveform" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Waveform, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="Header" SelectedValueMemberPath="DataFilePath"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:CalibrationConfiguratorView}}, Path=DataContext.ParentProject.AvailableWaveforms,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick" EditorStyle="{StaticResource DropDownOnCellEdit}">
</t:GridViewComboBoxColumn>
<t:GridViewDataColumn Header="Comment" Width="Auto" HeaderTextAlignment="Center"
IsVisible="False"
DataMemberBinding="{Binding Path=Comment, Mode=TwoWay}">
</t:GridViewDataColumn>
</t:RadGridView.Columns>
</t:RadGridView>

</Grid>
</t:GroupBox>

<TextBox x:Name="textBoxDebug" Grid.Row="1" Grid.ColumnSpan="2" Padding="10 0 0 0"
Text="{Binding Path=ConfigStatus, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
TextWrapping="Wrap" BorderBrush="DodgerBlue"  IsReadOnly="True"
VerticalContentAlignment="Center"
Height="30" Width="Auto" Background="Transparent" Foreground="White" FontSize="14"
Visibility="Collapsed">
</TextBox>

</Grid>
</UserControl>
```

LimitUcEditor.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Calibration.LimitUcEditor"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:custom="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.Calibration"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<Style TargetType="t:RadTreeViewItem" x:Key="FolderIconStyle">
<Setter Property="DefaultImageSrc" Value="FolderClosed_16x.png" />
<Setter Property="ExpandedImageSrc" Value="FolderOpened_16x.png" />
</Style>

<HierarchicalDataTemplate x:Key="RootCalData" >
<TextBlock Text="{Binding ResultName}" ToolTip="{Binding ResultName}" />
</HierarchicalDataTemplate>
</UserControl.Resources>

<Grid>
<Grid.Resources>
<Style x:Key="style1"
TargetType="t:GridViewCell">
<Setter Property="Background"
Value="{t:VisualStudio2013Resource ResourceKey=AccentBrush}" />
</Style>
</Grid.Resources>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu Grid.Row="0" >
<TextBlock Text="Limits Section:" ToolTip="Use the Combo Box to the right to change the limits
section displayed in the Grid View." FontSize="13"/>
<t:RadComboBox x:Name="SelectLimitSectionComboBox" t:ToolTip="Value of limit file section
(provides the data displayed below)."
Width="Auto" Background="DodgerBlue"
DisplayMemberPath="ResultName"  FontSize="13"
```

```
ItemsSource="{Binding Path=CalLimits.Results, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=SelectedCalResult, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">
</t:RadComboBox>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Generate Limits" Command="{Binding
AutoGenerateCalLimitsCommand}"  FontSize="13"
t:ToolTip="Calculates and sets the Min/Max/Tolerance values using the desired tolerence
parameters, for all limit items in the grid view."/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="Min (-):" ToolTip="Use the Numeric Box to the right to change the desired
Minimum Tolerance." FontSize="13" />
<t:RadNumericUpDown MinWidth="60" Value="{Binding Path=LimitMin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" Minimum="0"  SmallChange="1"
t:ToolTip="Value of desired Min Tolerance" FontSize="13"/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="Max (+):" ToolTip="Use the Numeric Box to the right to change the desired
Maximum Tolerance." FontSize="13"/>
<t:RadNumericUpDown MinWidth="60" Value="{Binding Path=LimitMax, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" Minimum="0"  SmallChange="1"
t:ToolTip="Value of desired Max Tolerance" FontSize="13" />
<t:RadMenuSeparatorItem/>
<t:RadComboBox x:Name="ViewingUnitComboBox"  ToolTip="Select a unit for viewing
frequency." IsEnabled="True"
SelectedValue="{Binding Path=ViewingUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=UnitList, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
SelectionChanged="ViewingUnitComboBox_SelectionChanged"/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=SelectedCalResult.KvPMergedObjectData.Count,
StringFormat={}{0} Entries, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of calibration limits in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="LimitEditorGridView" Grid.Row="1"
ItemsSource="{Binding Path=SelectedCalResult.KvPMergedObjectData, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
Height="Auto" Width="Auto"
AutoGenerateColumns ="True"
AutoGeneratingColumn="GridView_AutoGeneratingColumn"
```

```
EnableColumnVirtualization="True"
EnableRowVirtualization="True"
RowIndicatorVisibility="Visible"
LeftFrozenColumnCount="1"

GroupRenderMode="Flat"
RowHeight="30"
FontSize="14"
SelectionUnit="Mixed"
SelectionMode="Extended"
PastingCellClipboardContent="LimitEditorGridView_PastingCellClipboardContent"
PreparingCellForEdit="LimitEditorGridView_PreparingCellForEdit"
CellEditEnded="LimitEditorGridView_CellEditEnded"
CellValidating="LimitEditorGridView_CellValidating">

<t:RadGridView.Columns>
<custom:RowNumberColumn Header="#" Width="30" CellStyle="{StaticResource style1}" />
</t:RadGridView.Columns>

</t:RadGridView>

</Grid>
</UserControl>
```

ResultDataView.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.ResultDataView"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:custom="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>

</UserControl.Resources>

<Grid>
<Grid.Resources>
```

```
<Style x:Key="style1"
TargetType="t:GridViewCell">
<Setter Property="Background"
Value="{t:VisualStudio2013Resource ResourceKey=AccentBrush}" />
</Style>
</Grid.Resources>

<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu Grid.Row="0">
<t:RadMenuItem Header="Export" Click="RadMenuItem_Click" >
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe109;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadComboBox x:Name="ViewingUnitComboBox"  ToolTip="Select a unit for viewing
frequency." IsEnabled="True"
SelectedValue="{Binding Path=ViewingUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=UnitList, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
SelectionChanged="ViewingUnitComboBox_SelectionChanged"/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=GridViewData.Count, StringFormat={}{0} Entries,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of calibration results in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="CalResultGridView" Grid.Row="1"
ItemsSource="{Binding Path=GridViewData, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
Height="Auto" Width="Auto"
AutoGeneratingColumn="CalResultGridView_AutoGeneratingColumn"
AutoGenerateColumns ="True"
EditTriggers="None"
EnableColumnVirtualization="True"
EnableRowVirtualization="True"
RowIndicatorVisibility="Visible"
```

```
LeftFrozenColumnCount="1"

GroupRenderMode="Flat"
RowHeight="30"
FontSize="14"
SelectionUnit="Mixed"
SelectionMode="Extended">

<t:RadGridView.Columns>
<custom:RowNumberColumn Header="#" Width="30" CellStyle="{StaticResource style1}" />
</t:RadGridView.Columns>

</t:RadGridView>
</Grid>
</UserControl>
```

ProductConfigurationView.xaml

```
<UserControl x:Class="MT.TestStudio.GUI.User_Controls.ProductConfigurationView"
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:MT.TestStudio.GUI.ViewModels"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="1000" d:DesignWidth="1000">

<UserControl.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>

</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</UserControl.Resources>

<ScrollViewer VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Auto">

<Grid Background="#252526">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
```

```
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<WrapPanel >

<t:GroupBox Header="Flow Execution Settings" Grid.Column="0">
<!--Grid to hold the Flow Execution values-->
<Grid>
<!--Create a grid to hold the label and textbox-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>
<ColumnDefinition Width="150" />
<ColumnDefinition Width="50" />
<ColumnDefinition Width="150" />
<ColumnDefinition Width="20" />
</Grid.ColumnDefinitions>

<!--Add labels-->
<Label Grid.Row="0" Grid.Column="0" FontSize="12" FontWeight="Normal" Foreground="White"
Margin="0,10,0,10">Continue On Fail</Label>
<Label Grid.Row="1" Grid.Column="0" FontSize="12" FontWeight="Normal" Foreground="White"
Margin="0,10,0,10">Stop On Fail</Label>
<Label Grid.Row="2" Grid.Column="0" FontSize="12" FontWeight="Normal" Foreground="White"
```

```
Margin="0,10,0,10">Continue On All Fail</Label>
<Label Grid.Row="3" Grid.Column="0" FontSize="12" FontWeight="Normal" Foreground="White"
Margin="0,10,0,10">Stop On Alarm</Label>
<Label Grid.Row="4" Grid.Column="0" FontSize="12" FontWeight="Normal" Foreground="White"
Margin="0,10,0,10">Continue On Alarm</Label>
<Label Grid.Row="5" Grid.Column="0" FontSize="12" FontWeight="Normal" Foreground="White"
Margin="0,10,0,10">Gold Unit Enable/Disable</Label>


<t:RadToggleSwitchButton Grid.Row="0" Grid.Column="2" Margin="5"
UncheckedContent="OFF" CheckedContent="ON"
IsChecked="{Binding Path=ContinueOnFail, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
/>
<t:RadToggleSwitchButton Grid.Row="1" Grid.Column="2" Margin="5"
UncheckedContent="OFF" CheckedContent="ON"
IsChecked="{Binding Path=StopOnFail, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
/>
<t:RadToggleSwitchButton Grid.Row="2" Grid.Column="2" Margin="5"
UncheckedContent="OFF" CheckedContent="ON"
IsChecked="{Binding Path=ContinueOnAllFail, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
/>
<t:RadToggleSwitchButton Grid.Row="3" Grid.Column="2" Margin="5"
UncheckedContent="OFF" CheckedContent="ON"
IsChecked="{Binding Path=StopOnalarm, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
/>
<t:RadToggleSwitchButton Grid.Row="4" Grid.Column="2" Margin="5"
UncheckedContent="OFF" CheckedContent="ON"
IsChecked="{Binding Path=ContinueOnAlarm, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
/>
<t:RadToggleSwitchButton Grid.Row="5" Grid.Column="2" Margin="5"
UncheckedContent="OFF" CheckedContent="ON"
IsChecked="{Binding Path=GoldUnitEnable, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
/>

</Grid>
</t:GroupBox>
```

```
<t:GroupBox Header="User Calibration Settings" Grid.Column="1">
<!--Grid to hold the User Calibration values-->
<Grid >
<!--Create a grid to hold the label and textbox-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>
<ColumnDefinition Width="150" />
<ColumnDefinition Width="50" />
<ColumnDefinition Width="150" />
<ColumnDefinition Width="20" />
</Grid.ColumnDefinitions>

<!--Add labels-->
<Label Grid.Row="6" Grid.Column="0" Margin="0,10,0,10" FontSize="12" FontWeight="Normal"
Foreground="White">User Calibration</Label>
<Label Grid.Row="7" Grid.Column="0" Margin="0,10,0,10" FontSize="12" FontWeight="Normal"
Foreground="White">Calibration Expiration</Label>

<t:RadToggleSwitchButton Grid.Row="6" Grid.Column="2" Margin="5"
UncheckedContent="OFF" CheckedContent="ON"
IsChecked="{Binding Path=UserCalibrationEnabled, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
/>
```

```
<t:RadNumericUpDown  Grid.Row="7" Grid.Column="2"
HorizontalContentAlignment="Center" VerticalAlignment="Center"  Margin="5"
Minimum="0" Maximum="100"
Value="{Binding Path=CalibrationExpiration, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged }"
HideTrailingZeros="True"
/>

</Grid>
</t:GroupBox>

<t:GroupBox Header="Misc. Settings" Grid.Column="2">
<!--Grid to hold the Misc. values-->
<Grid>
<!--Create a grid to hold the label and textbox-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>
<ColumnDefinition Width="150" />
<ColumnDefinition Width="50" />
<ColumnDefinition Width="150" />
<ColumnDefinition Width="20" />
</Grid.ColumnDefinitions>

<!--Add labels-->
```

```
<Label Grid.Row="8" Grid.Column="0"  FontSize="12" FontWeight="Normal" Margin="0,10,0,10"
Foreground="White">Offline QA Enable</Label>
<Label Grid.Row="9" Grid.Column="0"  FontSize="12" FontWeight="Normal" Margin="0,10,0,10"
Foreground="White">Encrypt Required</Label>
<Label Grid.Row="10" Grid.Column="0"  FontSize="12" FontWeight="Normal" Margin="0,10,0,10"
Foreground="White">Inline enabled nth device</Label>
<Label Grid.Row="11" Grid.Column="0"  FontSize="12" FontWeight="Normal" Margin="0,10,0,10"
Foreground="White">Log nth Device</Label>
<Label Grid.Row="12" Grid.Column="0"  FontSize="12" FontWeight="Normal" Margin="0,10,0,10"
Foreground="White">Number of Sites</Label>

<t:RadToggleSwitchButton Grid.Row="8" Grid.Column="2" Margin="5"
UncheckedContent="OFF" CheckedContent="ON"
IsChecked="{Binding Path=OfflineQaAvailable, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
/>
<t:RadToggleSwitchButton Grid.Row="9" Grid.Column="2" Margin="5"
UncheckedContent="OFF" CheckedContent="ON"
IsChecked="{Binding Path=EncryptRequired, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
/>
<t:RadNumericUpDown  Grid.Row="10" Grid.Column="2"
HorizontalContentAlignment="Center" VerticalAlignment="Center"  Margin="5"
Minimum="0" Maximum="999"
Value="{Binding Path=InlineNthDevice, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged
}"
HideTrailingZeros="True"
/>
<t:RadNumericUpDown  Grid.Row="11" Grid.Column="2"
HorizontalContentAlignment="Center" VerticalAlignment="Center"  Margin="5"
Minimum="0" Maximum="999"
Value="{Binding Path=LogNthDevice, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged }"

HideTrailingZeros="True"
/>
<t:RadNumericUpDown  Grid.Row="12" Grid.Column="2"
HorizontalContentAlignment="Center" VerticalAlignment="Center"  Margin="5"
Minimum="0" Maximum="8"
Value="{Binding Path=NumberOfsites, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged
}"
HideTrailingZeros="True"
/>
```

```
</Grid>
</t:GroupBox>

</WrapPanel>
</Grid>

</ScrollViewer>

</UserControl>
```

ProjectConfiguration.xaml

```
<UserControl x:Class="MT.TestStudio.GUI.User_Controls.ProjectConfiguration"
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:MT.TestStudio.GUI.ViewModels"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="1000" d:DesignWidth="1000">
<UserControl.Resources>

<Style x:Key="TextBoxStyle" TargetType="TextBox">
<Style.Triggers>
<Trigger Property="IsEnabled" Value="True">
<Setter Property="Background" Value="Yellow"/>
</Trigger>
<Trigger Property="IsReadOnly" Value="True">
<Setter Property="Background" Value="Red"/>
</Trigger>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="IsEnabled" Value="True"/>
<Condition Property="IsReadOnly" Value="True"/>
</MultiTrigger.Conditions>
<Setter Property="Background" Value="Green"/>
</MultiTrigger>
</Style.Triggers>
</Style>

</UserControl.Resources>
```

```
<ScrollViewer VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Auto">
<Grid Background="#F0F0F0" Margin="0,-3,0,0">
<DockPanel LastChildFill="True">

<Label Content="Project Properties" Foreground="Black" FontWeight="ExtraBold" FontSize="24"
Margin="50,50,0,50" DockPanel.Dock="Top" />


<Grid DockPanel.Dock="Top" Margin="0,0,0,0">
<!--Create a grid to hold the label and textbox-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>
<ColumnDefinition Width="104" />
<ColumnDefinition Width="*" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0" Grid.Row="0" Margin="50,0,0,0" DockPanel.Dock="Top"
FontWeight="Bold" FontSize="12" Foreground="Black">General</Label>
<Separator Grid.Column="1" Grid.Row="0" Margin="0,4,270,0" Height="1"
Background="#6E6E6E" />
</Grid>


<!--Grid to hold the first four Cal Def values-->
<Grid DockPanel.Dock="Top" Margin="50,25,50,50">
<!--Create a grid to hold the label and textbox-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
```

```
<Grid.ColumnDefinitions>
<ColumnDefinition Width="150" />
<ColumnDefinition Width="50" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="20" />
</Grid.ColumnDefinitions>

<!--Add labels-->
<Label Margin="0,10,0,10" Grid.Row="0" Grid.Column="0" FontSize="12"
Foreground="Black">Product Name</Label>
<Label Margin="0,10,0,10" Grid.Row="1" Grid.Column="0" FontSize="12"
Foreground="Black">Revision</Label>
<Label Margin="0,10,0,10" Grid.Row="2" Grid.Column="0" FontSize="12"
Foreground="Black">Project File Name</Label>
<Label Margin="0,10,0,10" Grid.Row="3" Grid.Column="0" FontSize="12"
Foreground="Black">Project File Path</Label>
<Label Margin="0,10,0,10" Grid.Row="4" Grid.Column="0" FontSize="12"
Foreground="Black">Project Creation Date</Label>
<Label Margin="0,10,0,10" Grid.Row="5" Grid.Column="0" FontSize="12"
Foreground="Black">Target System</Label>

<!--Add Bound TextBoxes-->
<TextBox Style="{StaticResource TextBoxStyle}" Margin="0,10,200,10" Padding="5,2,5,0"
Background="White" Foreground="Black" Grid.Row="0" Grid.Column="2" FontSize="12"
IsReadOnly="True" Text="{Binding Path=PVM.ParentProject.ProductName, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" />
<TextBox Style="{StaticResource TextBoxStyle}" Margin="0,10,200,10" Padding="5,2,5,0"
Background="White" Foreground="Black" Grid.Row="1" Grid.Column="2" FontSize="12"
IsReadOnly="True" Text="{Binding Path=PVM.ParentProject.Version, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" />
<TextBox Style="{StaticResource TextBoxStyle}" Margin="0,10,200,10" Padding="5,2,5,0"
Background="White" Foreground="Black" Grid.Row="2" Grid.Column="2" FontSize="12"
IsReadOnly="True" Text="{Binding Path=PVM.ParentProject.ProjectName, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" />
<TextBox Style="{StaticResource TextBoxStyle}" Margin="0,10,200,10" Padding="5,2,5,0"
Background="White" Foreground="Black" Grid.Row="3" Grid.Column="2" FontSize="12"
IsReadOnly="True" Text="{Binding Path=PVM.ParentProject.ProjectFilePath, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" />
<TextBox Style="{StaticResource TextBoxStyle}" Margin="0,10,200,10" Padding="5,2,5,0"
Background="White" Foreground="Black" Grid.Row="4" Grid.Column="2" FontSize="12"
IsReadOnly="True" Text="N/A"/>
```

```
<t:RadComboBox Margin="0,10,200,10"  Grid.Row="5" Grid.Column="2" FontSize="12" />
<t:RadComboBox x:Name="UTS_ComboBox"  Margin="0,10,200,10"  Grid.Row="5"
Grid.Column="2" FontSize="12" ToolTip="Target system for this project."
SelectedItem="{Binding Path=PVM.ParentProject.UserTargetSystem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=TargetSystemOptions, Mode=OneTime}"
SelectionChanged="RadComboBox_SelectionChanged">
</t:RadComboBox>
</Grid>

<Grid DockPanel.Dock="Top" Margin="0,0,0,0">
<!--Create a grid to hold the label and textbox-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>
<ColumnDefinition Width="120" />
<ColumnDefinition Width="*" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0" Grid.Row="0" Margin="50,0,0,0" DockPanel.Dock="Top"
FontWeight="Bold" FontSize="12" Foreground="Black">Meta Data</Label>
<Separator Grid.Column="1" Grid.Row="0" Margin="0,4,270,0" Height="1"
Background="#6E6E6E" />
</Grid>



<Grid DockPanel.Dock="Top" Margin="50,25,50,50">
<!--Create a grid to hold the label and textbox-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
```

```
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>
<ColumnDefinition Width="150" />
<ColumnDefinition Width="50" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="20" />
</Grid.ColumnDefinitions>

<!--Add labels-->
<Label Margin="0,10,0,10" Grid.Row="0" Grid.Column="0" FontSize="12" FontWeight="Normal"
Foreground="Black">Project Name</Label>
<Label Margin="0,10,0,10" Grid.Row="1" Grid.Column="0" FontSize="12" FontWeight="Normal"
Foreground="Black">Comment</Label>

<TextBox Margin="0,10,200,10" Padding="5,0,5,0" Background="White" Foreground="Black"
Grid.Row="0" Grid.Column="2" FontSize="12" Text="{Binding
Path=PVM.ParentProject.ProjectName, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" />
<TextBox Margin="0,10,200,10"
Padding="5,0,5,0"
Background="White"
Foreground="Black"
Grid.Row="1" Grid.Column="2"
FontSize="12"
Text="N/A"
TextWrapping="Wrap"
AcceptsReturn="True"
SpellCheck.IsEnabled="True"
MinLines="6"/>

</Grid>
</DockPanel>
</Grid>
</ScrollViewer>
</UserControl>
```

SystemConfiguration.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.SystemConfiguration"
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:MerlinTestStudio_Demo_Telerik"
mc:Ignorable="d"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.InputBindings>
<KeyBinding Key="Delete" Command="{Binding DeleteInstrumentCommand}"/>
</UserControl.InputBindings>

<UserControl.Resources>

</UserControl.Resources>

<Grid>

<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="0.5*"/>
<ColumnDefinition Width="3*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<t:RadMenu Grid.Row="0" Grid.ColumnSpan="3">
<t:RadMenuItem  Header="Auto Search Instruments" IsEnabled="False">
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe13b;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Reset System" Command="{Binding ResetSystemCommand}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Delete Instrument" Command="{Binding
DeleteInstrumentCommand}"/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:GroupBox Header="Add Instruments" Grid.Row="1" Grid.Column="0">
<Grid >
```

```
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="*"/>
<RowDefinition Height="*"/>
<RowDefinition Height="*"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
<t:RadButton Grid.Row="0" Command="{Binding Path=AddInstrumentCommand}"
Content="RF110" CommandParameter="RF110"/>
<t:RadButton Grid.Row="1" Command="{Binding Path=AddInstrumentCommand}"
Content="RF210" CommandParameter="RF210"/>
<t:RadButton Grid.Row="2" Command="{Binding Path=AddInstrumentCommand}"
Content="PE32H" CommandParameter="PE32H"/>
<t:RadButton Grid.Row="3" Command="{Binding Path=AddInstrumentCommand}"
Content="SEDPin32" CommandParameter="SEDPin32"/>
<t:RadButton Grid.Row="4" Command="{Binding Path=AddInstrumentCommand}"
Content="PSM" CommandParameter="PSM"/>
</Grid>
</t:GroupBox>

<t:GroupBox Header="Configuration" Grid.Row="1" Grid.Column="2" >
<t:RadListBox x:Name="InstrumentListDisplay"
Width="Auto"
ItemsSource="{Binding Path=Instruments, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=Selected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
>
<t:RadListBox.ItemTemplate>
<DataTemplate >
<TextBlock Text="{Binding Path=InstrumentName}" FontSize="14" />
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>
</t:GroupBox>


<t:GroupBox Header="Instrument Panel" Grid.Row="1" Grid.Column="1">
<Grid >
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

```
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<!-- Instrument property UI elements here-->
<Grid Grid.Row="0" Grid.Column="1" VerticalAlignment="Top" HorizontalAlignment="Left">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<TextBlock Grid.Row="0" Text="Instrument Name: " Margin="0 10 0 0" />
<TextBox Grid.Row="1" Text="{Binding Path=instrumentName, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
<TextBlock Grid.Row="2" Text="Instrument ID: " Margin="0 10 0 0"/>
<TextBox Grid.Row="3" Text="{Binding Path=instrumentID, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
<TextBlock Grid.Row="4" Text="Instrument SN: " Margin="0 10 0 0"/>
<TextBox Grid.Row="5" Text="{Binding Path=serialNumber, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
</Grid>
<StackPanel Grid.Row="0" Grid.Column="0" HorizontalAlignment="Center"
VerticalAlignment="Center" Visibility="Collapsed" >
<WrapPanel Margin="5">
<TextBlock Text="Instrument Name: "/>
<TextBox Text="{Binding Path=instrumentName, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
</WrapPanel>
<WrapPanel Margin="5">
<TextBlock Text="Instrument ID: "/>
<TextBox Text="{Binding Path=instrumentID, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
</WrapPanel>
<WrapPanel Margin="5">
<TextBlock Text="Instrument SN: "/>
<TextBox Text="{Binding Path=serialNumber, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
```

```
</WrapPanel>
</StackPanel>

<t:GroupBox Header="Ports" Grid.Row="0" Grid.Column="2" Width="Auto">
<t:RadListBox ItemsSource="{Binding Path=ports, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Width="Auto" >
</t:RadListBox>
</t:GroupBox>
</Grid>
</t:GroupBox>

</Grid>
</UserControl>
```

CommonResources.xaml

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:swimlane="clr-namespace:MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:telerik="http://schemas.telerik.com/2008/xaml/presentation"
>
<telerik:BooleanToVisibilityConverter x:Key="booleanToVisibility" />
<Style x:Key="addButtonStyle"
BasedOn="{StaticResource RadButtonStyle}"
TargetType="telerik:RadButton">
<Setter Property="Background" Value="#FF333333" />
<Setter Property="UseLayoutRounding" Value="True" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="telerik:RadButton">
<Grid>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="CommonStates">
<VisualState x:Name="Normal" />
<VisualState x:Name="MouseOver">
<Storyboard>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="OuterMouseOverBorder"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="00:00:00">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
```

```
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Pressed">
<Storyboard>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="OuterPressedBorder"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="00:00:00">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
<Storyboard>
<DoubleAnimation Duration="0"
Storyboard.TargetName="OuterBorder"
Storyboard.TargetProperty="Opacity"
To="0.5" />
<DoubleAnimation Duration="0"
Storyboard.TargetName="Content"
Storyboard.TargetProperty="Opacity">
<DoubleAnimation.To>0.3</DoubleAnimation.To>
</DoubleAnimation>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="BackgroundVisibility">
<VisualState x:Name="BackgroundIsHidden" />
<VisualState x:Name="BackgroundIsVisible" />
</VisualStateGroup>
<VisualStateGroup x:Name="FocusStatesGroup">
<VisualState x:Name="Unfocused" />
<VisualState x:Name="Focused" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<!--  normal  -->
<Border x:Name="OuterBorder"
```

```
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
CornerRadius="{TemplateBinding CornerRadius}" />

<!--  mouseover  -->
<Border x:Name="OuterMouseOverBorder"
Background="#FF309B46"
CornerRadius="{TemplateBinding CornerRadius}"
Visibility="Collapsed" />

<!--  pressed  -->
<Border x:Name="OuterPressedBorder"
Background="#FF84A48B"
CornerRadius="{TemplateBinding CornerRadius}"
Visibility="Collapsed" />

<ContentControl x:Name="Content"
Margin="{TemplateBinding Padding}"
HorizontalContentAlignment="{TemplateBinding HorizontalContentAlignment}"
VerticalContentAlignment="{TemplateBinding VerticalContentAlignment}"
Content="{TemplateBinding Content}"
ContentTemplate="{TemplateBinding ContentTemplate}"
Foreground="{TemplateBinding Foreground}"
IsTabStop="False" />
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="removeButtonStyle"
BasedOn="{StaticResource RadButtonStyle}"
TargetType="telerik:RadButton">
<Setter Property="Background" Value="#FF333333" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="telerik:RadButton">
<Grid>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="CommonStates">
<VisualState x:Name="Normal" />
```

```
<VisualState x:Name="MouseOver">
<Storyboard>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="OuterMouseOverBorder"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="00:00:00">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Pressed">
<Storyboard>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="OuterPressedBorder"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="00:00:00">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
<Storyboard>
<DoubleAnimation Duration="0"
Storyboard.TargetName="OuterBorder"
Storyboard.TargetProperty="Opacity"
To="0.5" />
<DoubleAnimation Duration="0"
Storyboard.TargetName="Content"
Storyboard.TargetProperty="Opacity">
<DoubleAnimation.To>0.3</DoubleAnimation.To>
</DoubleAnimation>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="BackgroundVisibility">
<VisualState x:Name="BackgroundIsHidden" />
<VisualState x:Name="BackgroundIsVisible" />
</VisualStateGroup>
```

```
<VisualStateGroup x:Name="FocusStatesGroup">
<VisualState x:Name="Unfocused" />
<VisualState x:Name="Focused" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<!--  normal  -->
<Border x:Name="OuterBorder"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
CornerRadius="{TemplateBinding CornerRadius}" />

<!--  mouseover  -->
<Border x:Name="OuterMouseOverBorder"
Background="#FFE22F07"
CornerRadius="{TemplateBinding CornerRadius}"
Visibility="Collapsed" />

<!--  pressed  -->
<Border x:Name="OuterPressedBorder"
Background="#FFDE9D8E"
CornerRadius="{TemplateBinding CornerRadius}"
Visibility="Collapsed" />

<ContentControl x:Name="Content"
Margin="{TemplateBinding Padding}"
HorizontalContentAlignment="{TemplateBinding HorizontalContentAlignment}"
VerticalContentAlignment="{TemplateBinding VerticalContentAlignment}"
Content="{TemplateBinding Content}"
ContentTemplate="{TemplateBinding ContentTemplate}"
Foreground="{TemplateBinding Foreground}"
IsTabStop="False" />
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="additionalContentStyle" TargetType="swimlane:AdditionalContent">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="swimlane:AdditionalContent">
```

```
<Grid>
<telerik:SettingsPane x:Name="settingsPane"
IsActive="False"
Diagram="{TemplateBinding Diagram}"
Visibility="Collapsed" />
<Canvas Margin="0 -75 -3 0"
HorizontalAlignment="Right"
VerticalAlignment="Center">
<StackPanel x:Name="addRemoveButtons" Visibility="Collapsed" Orientation="Horizontal">
<telerik:RadButton Width="18"
Height="18"
Margin="0 0 6 0"
Background="#FF333333"
Command="swimlane:SwimlaneCommands.RemoveCommand"
CornerRadius="2"
Style="{StaticResource removeButtonStyle}">
<Rectangle Width="10"
Height="2"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Fill="White" />
</telerik:RadButton>
<telerik:RadButton Width="18"
Height="18"
Background="#FF333333"
Command="swimlane:SwimlaneCommands.AddCommand"
CornerRadius="2"
Style="{StaticResource addButtonStyle}">
<Path Width="10"
Height="10"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M4,0 L6,0 L6,4 L10,4 L10,6 L6,6 L6,10 L4,10 L4,6 L0,6 L0,4 L4,4 z"
Fill="White"
Stretch="Fill"
UseLayoutRounding="False" />
</telerik:RadButton>
</StackPanel>
</Canvas>
</Grid>
</ControlTemplate>
</Setter.Value>
```

```
</Setter>
</Style>

<Style BasedOn="{StaticResource additionalContentStyle}"
TargetType="swimlane:AdditionalContent" />
</ResourceDictionary>
```

ConnectionDiagram.xaml

```
<UserControl x:Class="MT.TestStudio.GUI.User_Controls.ConnectionDiagram"
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:MT.TestStudio.GUI.ViewModels"
xmlns:converters="clr-namespace:MerlinTestStudio_Demo_Telerik.Converters"
mc:Ignorable="d"
d:DesignHeight="300" d:DesignWidth="300">
<UserControl.DataContext>
<!--Declaratively create an instance of our View Model-->
<local:ConnectionDiagramViewModel/>

</UserControl.DataContext>
<UserControl.Resources>
<converters:UriToCachedImageConverter x:Key="uriToImageConv" />
</UserControl.Resources>

<Grid Background="#252526">

<DockPanel LastChildFill="True">

<Image Source="{Binding DisplayedImage, Converter={StaticResource uriToImageConv}}"
HorizontalAlignment="Left" Margin="20" Name="image1" Stretch="Fill"
VerticalAlignment="Bottom" />

</DockPanel>
</Grid>

</UserControl>
```

Digital_Cable_Map.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager.Digital_Cable_Map"
```

```
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"
mc:Ignorable="d"
xmlns:diagramsCore="clr-
namespace:Telerik.Windows.Diagrams.Core;assembly=Telerik.Windows.Diagrams.Core"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:primitives="clr-
namespace:Telerik.Windows.Controls.Diagrams.Primitives;assembly=Telerik.Windows.Controls.D
iagrams"
x:Name="root"
HorizontalAlignment="Stretch"
VerticalAlignment="Stretch"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary Source="Resources.xaml" />
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</UserControl.Resources>

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="66" />
<RowDefinition />
</Grid.RowDefinitions>
<local:SqlDiagram x:Name="diagram"
Grid.Row="1"
Margin="0 5 0 0"
BorderBrush="#d6d4d4"
BorderThickness="1"
Deserialized="OnDiagramDeserialized"
ConnectionManipulationCompleted="OnConnectionManipulationCompleted"
ConnectionRoundedCorners="True"
GraphSource="{Binding}"
ItemsChanging="OnItemsChanging"
IsBackgroundSurfaceVisible="False"
IsSnapToGridEnabled="False"
```

```
IsSnapToItemsEnabled="False"
ShapeStyleSelector="{StaticResource styleSelector}"
ScrollViewer.HorizontalScrollBarVisibility="Auto"
ScrollViewer.VerticalScrollBarVisibility="Auto"
PreviewSelectionChanged="OnPreviewSelectionChanged"
RouteConnections="True"
SelectionChanged="OnSelectionChanged">
<primitives:ItemInformationAdorner.AdditionalContent>
<local:TableAdditionalContent ContextItem="{Binding SelectedItem, ElementName=diagram}"
Diagram="{Binding ElementName=diagram}" />
</primitives:ItemInformationAdorner.AdditionalContent>
</local:SqlDiagram>

<Border BorderBrush="#d6d4d4" BorderThickness="1">
<Border.Resources>
<Style BasedOn="{StaticResource RadGeometryButtonStyle}"
TargetType="t:RadGeometryButton">
<Setter Property="BorderThickness" Value="1" />
<Setter Property="Foreground" Value="#FF5E5E5E" />
<Setter Property="t:GeometryButtons.GeometryFill" Value="#FF434647" />
<Setter Property="VerticalAlignment" Value="Center" />
<Setter Property="Margin" Value="0" />
</Style>
</Border.Resources>
<Border VerticalAlignment="Center" Padding="24,0,24,0">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
<ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<t:RadGeometryButton Click="OnNewClick"
Content="new"
Geometry="M11,0 L18,7 L11,7 z M0,0 L10,0 L10,7 L10,8 L18,8 L18,21 L0,21 z"
ToolTipService.ToolTip="new" />
<t:RadGeometryButton Grid.Column="1"
Command="t:DiagramCommands.Open"
Content="open"
Geometry="M10,0L17,0C17.5523,1.19209e-007 18,0.447715 18,1L18.5,3L21,3C21.5523,3
22,3.44772 22,4L22,18C22,18.5523 21.5523,19 21,19L1,19C0.447715,19 0,18.5523
```

```
0,18L0,4C0,3.44772 0.447715,3 1,3L8.5,3L9,1C9,0.447715 9.44771,1.19209e-007 10,0z"
ToolTipService.ToolTip="open" />
<t:RadGeometryButton Grid.Column="2"
Command="t:DiagramCommands.Save"
Content="save"
Geometry="F1 M0,0 L22,0 L22,22 L16,22 L16,15 L6,15 L6,22 L0,22 L0,0 z M8,22 L8,19 L12,19
L12,22 L8,22 z M5,2 L5,8 L18,8 L18,2 L5,2 z"
ToolTipService.ToolTip="save" />

<t:RadGeometryButton Grid.Column="3"
Command="t:DiagramCommands.Undo"
CommandTarget="{Binding ElementName=diagram}"
Content="undo"
Geometry="M4.2667065,0 L8.4021349,0 L5.2587953,4.1229997 L13.039669,4.1229997
C16.905663,4.1229997 20.039669,7.2570057 20.039669,11.122999 C20.039669,14.988993
16.905663,18.122999 13.039669,18.122999 L11.825022,18.122999 L11.825022,14.123001
L13.039669,14.123001 C14.696525,14.123001 16.039671,12.779856 16.039671,11.123001
C16.039671,9.4661465 14.696525,8.1230011 13.039669,8.1230011 L5.2947307,8.1230011
L8.4021349,12.248009 L4.2667065,12.248009 L0,5.9990273 z"
ToolTipService.ToolTip="undo"
t:GeometryButtons.EllipseHeight="40"
t:GeometryButtons.EllipseWidth="40" />
<t:RadGeometryButton Grid.Column="4"
Command="t:DiagramCommands.Redo"
CommandTarget="{Binding ElementName=diagram}"
Content="redo"
Geometry="M12,0 L16,0 L20,6 L16,12 L12,12 L14.666667,8 L7,8 C5.3431458,8 4,9.3431454 4,11
C4,12.656855 5.3431458,14 7,14 L8,14 L8,18 L7,18 C3.1340067,18 0,14.865993 0,11
C0,7.134007 3.1340067,4 7,4 L14.666667,4 z"
ToolTipService.ToolTip="redo" />
</Grid>
</Border>
</Border>
</Grid>
</UserControl>
```

PinMap.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager.PinMap"
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:MerlinTestStudio_Demo_Telerik"
xmlns:conv="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Converters"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.InputBindings>
<KeyBinding Key="Esc" Command="{Binding ClearRowSelectionCommand}"/>
</UserControl.InputBindings>

<UserControl.Resources>

</UserControl.Resources>

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Pin Map Toolbar-->
<t:RadMenu Grid.Row="0" >
<t:RadMenuItem x:Name="AddPinBtn" Header="Pin"  ToolTip="Add a pin" Command="{Binding
Path=AddPinCommand}">
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe11e;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem x:Name="RemovePinBtn" Header="Pin"  ToolTip="Remove a pin"
Command="{Binding Path=RemovePinCommand}">
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe10c;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem />
<t:RadMenuItem x:Name="ImportPinsBtn" Header="Generate Pins"  ToolTip="Imports Multiple
Pins" Command="{Binding Path=ImportPinsCommand}" IsEnabled="False">
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe63e;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
```

```
<t:RadMenuSeparatorItem />
<t:RadMenuItem x:Name="AddSiteBtn" Header="Site"  ToolTip="Add a site"
Command="{Binding Path=AddSiteCommand}">
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe11e;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem x:Name="RemoveSiteBtn" Header="Site"  ToolTip="Remove a site"
Command="{Binding Path=ShowRemoveSiteWindowCommand}" >
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe10c;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem />
<TextBlock Text="{Binding ElementName=PinMapGrid, Path=Items.Count, StringFormat={}{0}
Entries, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of time sets in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<Grid Grid.Row="1">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<t:RadListBox x:Name="PinMapSectionListBox" Grid.Column="0"
SelectedItem="{Binding Path=SelectedPinMapSection, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
>
<t:RadListBox.Resources>
<Style TargetType="t:RadListBoxItem">
<Setter Property="Height" Value="50"/>
<Setter Property="VerticalContentAlignment" Value="Center"/>
<Setter Property="HorizontalContentAlignment" Value="Center"/>
</Style>
</t:RadListBox.Resources>
</t:RadListBox>

<t:RadGridView x:Name="PinMapGrid" Grid.Column="1"
AutoGenerateColumns = "False"
CanUserFreezeColumns = "False"
```

```
RowIndicatorVisibility = "Visible"
ShowGroupPanel = "False"
CanUserSortColumns = "False"
IsFilteringAllowed = "False"

GroupRenderMode = "Flat"
EnableColumnVirtualization = "True"
EnableRowVirtualization = "True"
LeftFrozenColumnCount = "2"
RowStyle = "{StaticResource PinMapGridViewRowStyle}"

RowHeight="30"
AllowDrop="False"
CanUserDeleteRows="True"
SelectionUnit="Mixed"
SelectionMode="Extended"
CellEditEnded="PinMapGrid_CellEditEnded"
PreparedCellForEdit="PinMapGrid_PreparedCellForEdit"
CellValidating="PinMapGrid_CellValidating"
>

<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>
</t:RadGridView>
</Grid>



</Grid>
</UserControl>
```

Resources.xaml

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:Primitives="clr-
namespace:Telerik.Windows.Controls.Diagrams.Primitives;assembly=Telerik.Windows.Controls.D
iagrams"
xmlns:diagramsCore="clr-
namespace:Telerik.Windows.Diagrams.Core;assembly=Telerik.Windows.Diagrams.Core"
```

```
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles"
xmlns:telerik="http://schemas.telerik.com/2008/xaml/presentation">

<ResourceDictionary.MergedDictionaries>
<ResourceDictionary Source="CommonResources.xaml" />
</ResourceDictionary.MergedDictionaries>

<Style BasedOn="{StaticResource additionalContentStyle}"
TargetType="local:TableAdditionalContent" />

<diagramsCore:ConnectorCollection x:Key="customConnectors">
<telerik:RadDiagramConnector x:Name="Left" Offset="0 0.5" />
<telerik:RadDiagramConnector x:Name="Right" Offset="1 0.5" />
<telerik:RadDiagramConnector x:Name="Auto"
Opacity="0"
Offset="0.5 0.5" />
</diagramsCore:ConnectorCollection>

<Style x:Key="expandCollapseButtonClassDiagramItemStyle"
TargetType="telerik:RadToggleButton">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="telerik:RadToggleButton">
<Grid Background="Transparent" Cursor="Hand">
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="CommonStates">
<VisualState x:Name="Normal" />
<VisualState x:Name="MouseOver">
<Storyboard>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect"
Storyboard.TargetProperty="Fill">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<SolidColorBrush Color="#59000000" />
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Pressed" />
</VisualStateGroup>
<VisualStateGroup x:Name="CheckStates">
```

```
<VisualState x:Name="Checked">
<Storyboard>
<DoubleAnimation Duration="0:0:0.2"
Storyboard.TargetName="arrow"
Storyboard.TargetProperty="(FrameworkElement.RenderTransform).Angle"
To="0" />
</Storyboard>
</VisualState>
<VisualState x:Name="Unchecked">
<Storyboard>
<DoubleAnimation Duration="0:0:0.2"
Storyboard.TargetName="arrow"
Storyboard.TargetProperty="(FrameworkElement.RenderTransform).Angle"
To="180" />
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Rectangle x:Name="rect"
Width="14"
Height="14"
Fill="#FF333333"
RadiusX="2"
RadiusY="2"
StrokeThickness="2" />
<Grid x:Name="arrow"
Width="14"
Height="14"
RenderTransformOrigin="0.5,0.5">
<Path x:Name="path1"
Width="6"
Height="4"
Margin="0,0,0,4"
HorizontalAlignment="Center"
VerticalAlignment="Center"
Data="M8,0 L8,1.999831 L4,4.2555118 L0,1.9998311 L0,1.1920929E-07 L4,2.255681 z"
Fill="White"
Stretch="Fill"
StrokeThickness="2"
UseLayoutRounding="False" />
<Path x:Name="path2"
Width="6"
```

```
Height="4"
Margin="0,0,0,3"
HorizontalAlignment="Center"
VerticalAlignment="Bottom"
Data="M8,0 L8,1.999831 L4,4.2555118 L0,1.9998311 L0,1.1920929E-07 L4,2.255681 z"
Fill="White"
Stretch="Fill"
StrokeThickness="2"
UseLayoutRounding="False" />
<Grid.RenderTransform>
<RotateTransform Angle="180" />
</Grid.RenderTransform>
</Grid>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="tableStyle" TargetType="local:TableShape">
<Setter Property="IsEditable" Value="True" />
<Setter Property="IsResizingEnabled" Value="False" />
<Setter Property="Foreground" Value="White" />
<Setter Property="HorizontalContentAlignment" Value="Stretch" />
<Setter Property="VerticalContentAlignment" Value="Center" />
<Setter Property="IsConnectorsManipulationEnabled" Value="False" />
<Setter Property="IsCollapsible" Value="True" />
<Setter Property="Padding" Value="0" />
<Setter Property="IsCollapsed" Value="{Binding IsCollapsed, Mode=TwoWay}" />
<Setter Property="Height" Value="{Binding Height, Mode=TwoWay}" />
<Setter Property="Background" Value="White" />
<Setter Property="BorderBrush" Value="#FF32B0EA" />
<Setter Property="BorderThickness" Value="1" />
<Setter Property="FontFamily" Value="Segoe UI" />
<Setter Property="Position" Value="{Binding Position, Mode=TwoWay}" />
<Setter Property="ContentTemplate">
<Setter.Value>
<DataTemplate>
<Border Margin="4 0 0 0" Padding="5 0">
<TextBlock FontFamily="Segoe UI"
FontSize="14"
Foreground="White"
```

```
Text="{Binding Content}" />
</Border>
</DataTemplate>
</Setter.Value>
</Setter>
<Setter Property="EditTemplate">
<Setter.Value>
<DataTemplate>
<Border Padding="5 0">
<TextBox FontFamily="Segoe UI"
FontSize="14"
Text="{Binding Content,
Mode=TwoWay}" />
</Border>
</DataTemplate>
</Setter.Value>
</Setter>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="local:TableShape">
<Grid x:Name="RootPanel">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="40" />
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="MouseStates">
<VisualState x:Name="Normal" />
<VisualState x:Name="MouseOver" />
</VisualStateGroup>
<VisualStateGroup x:Name="CollapsedStates">
<VisualState x:Name="Expanded" />
<VisualState x:Name="Collapsed" />
</VisualStateGroup>
<VisualStateGroup x:Name="ActiveConectionStates">
<VisualState x:Name="NormalActiveConnectionState" />
<VisualState x:Name="ActiveConnectionInsideShape">
<Storyboard>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="ActiveSelectedBorder"
```

```
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="ConnectorsAdornerVisibilityStates">
<VisualState x:Name="ConnectorsAdornerCollapsed" />
<VisualState x:Name="ConnectorsAdornerVisible">
<Storyboard>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="ConnectorsControl"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="SelectionStates">
<VisualState x:Name="Selected" />
<VisualState x:Name="SelectedInGroup">
<Storyboard>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="SelectedBorder"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Unselected" />
```

```
<VisualState x:Name="SelectedAsGroup" />
</VisualStateGroup>
<VisualStateGroup x:Name="EditMode">
<VisualState x:Name="NormalMode" />
<VisualState x:Name="NormalEditMode">
<Storyboard>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="NormalContent"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Collapsed</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="EditContent"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="TextBoxEditMode" />
</VisualStateGroup>
<VisualStateGroup x:Name="DropStates">
<VisualState x:Name="DropNormal" />
<VisualState x:Name="DropComplete" />
<VisualState x:Name="DragOver">
<Storyboard>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="DragOverBorder"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
```

```
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="ContainerBorder"
Grid.RowSpan="4"
Background="{TemplateBinding Background}"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}" />
<Border x:Name="SelectedBorder"
Grid.RowSpan="4"
BorderBrush="#FFADD6FF"
BorderThickness="1"
Visibility="Collapsed" />
<Border x:Name="DragOverBorder"
Grid.RowSpan="4"
Margin="-4"
BorderBrush="#7FC92931"
BorderThickness="4"
Visibility="Collapsed" />
<Border x:Name="ActiveSelectedBorder"
Grid.RowSpan="4"
BorderBrush="#7FC92931"
BorderThickness="2"
Visibility="Collapsed" />
<Grid Height="50"
Margin="1"
Background="#FF32B0EA">
<Grid.ColumnDefinitions>
<ColumnDefinition />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>
<ContentPresenter x:Name="NormalContent"
Margin="{TemplateBinding Padding}"
HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
Content="{TemplateBinding Content}"
ContentTemplate="{TemplateBinding ContentTemplate}"
telerik:DiagramBehaviors.TextWrapping="Wrap" />
<Grid x:Name="PART_RotationalPart">
<ContentPresenter x:Name="EditContent"
HorizontalAlignment="Left"
```

```
VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
Content="{TemplateBinding Content}"
ContentTemplate="{TemplateBinding EditTemplate}"
Visibility="Collapsed" />
</Grid>
<telerik:RadToggleButton x:Name="ToggleCollapseButton"
Grid.Column="1"
Width="18"
MinHeight="18"
Margin="3 3 8 3"
VerticalAlignment="Center"
InnerCornerRadius="0"
IsBackgroundVisible="False"
IsTabStop="False"
Padding="3"
Style="{StaticResource expandCollapseButtonClassDiagramItemStyle}" />
</Grid>
<Grid Grid.Row="1"
Margin="1"
VerticalAlignment="Stretch"
Background="White">
<Grid.ColumnDefinitions>
<ColumnDefinition />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock Margin="10 0 0 0"
HorizontalAlignment="Left"
VerticalAlignment="Center"
FontFamily="Segoe UI"
FontSize="12"
Foreground="#FF787878"
Text="Coumn Name" />
<TextBlock Grid.Column="1"
HorizontalAlignment="Left"
VerticalAlignment="Center"
FontFamily="Segoe UI"
FontSize="12"
Foreground="#FF787878"
Text="Data Type" />
</Grid>
<ContentControl x:Name="CollapsedContent"
Grid.Row="2"
```

```
Margin="1"
HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
Background="White"
Content="{TemplateBinding CollapsedContent}"
ContentTemplate="{TemplateBinding CollapsedContentTemplate}">
<ContentControl.Visibility>
<Binding Path="IsCollapsed" RelativeSource="{RelativeSource TemplatedParent}">
<Binding.Converter>
<telerik:BooleanToVisibilityConverter />
</Binding.Converter>
</Binding>
</ContentControl.Visibility>
</ContentControl>
<Primitives:ConnectorsControl x:Name="ConnectorsControl"
Grid.RowSpan="4"
ItemContainerStyle="{TemplateBinding ConnectorStyle}"
Visibility="Collapsed" />
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="rowStyle" TargetType="local:RowShape">
<Setter Property="Background" Value="Transparent" />
<Setter Property="IsDraggingEnabled" Value="false" />
<Setter Property="IsResizingEnabled" Value="false" />
<Setter Property="IsRotationEnabled" Value="false" />
<Setter Property="local:AttachedProperties.CustomConnectors" Value="{StaticResource
customConnectors}" />
<!--  <Setter Property="telerik:DragDropManager.AllowDrag" Value="True" />  -->
<Setter Property="HorizontalContentAlignment" Value="Stretch" />
<Setter Property="VerticalContentAlignment" Value="Center" />
<Setter Property="Width" Value="240" />
<Setter Property="Height" Value="30" />
<Setter Property="Position" Value="{Binding Position, Mode=TwoWay}" />
<Setter Property="ContentTemplate">
<Setter.Value>
<DataTemplate>
<Border>
<Grid Margin="10 0"
```

```
HorizontalAlignment="Stretch"
VerticalAlignment="Center">
<Grid.ColumnDefinitions>
<ColumnDefinition />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBlock FontFamily="Segoe UI"
FontSize="11"
Foreground="#FF161616"
Text="{Binding ColumnName}" />
<TextBlock Grid.Column="1"
FontFamily="Segoe UI"
FontSize="11"
Foreground="#FF161616"
Text="{Binding DataType}" />
</Grid>
</Border>
</DataTemplate>
</Setter.Value>
</Setter>
<Setter Property="EditTemplate">
<Setter.Value>
<DataTemplate>
<Border HorizontalAlignment="Stretch" Background="Transparent">
<Grid Margin="10 0"
HorizontalAlignment="Stretch"
VerticalAlignment="Center">
<Grid.ColumnDefinitions>
<ColumnDefinition />
<ColumnDefinition />
</Grid.ColumnDefinitions>
<TextBox Margin="0 0 10 0"
HorizontalAlignment="Stretch"
Text="{Binding ColumnName,
Mode=TwoWay}" />
<telerik:RadComboBox Grid.Column="1"
ItemsSource="{local:DataTypeExtension}"
SelectedItem="{Binding DataType,
Mode=TwoWay}" />
</Grid>
</Border>
</DataTemplate>
```

```
</Setter.Value>
</Setter>
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="local:RowShape">
<Grid Background="{TemplateBinding Background}">
<VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="MouseStates">
<VisualState x:Name="Normal" />
<VisualState x:Name="MouseOver" />
</VisualStateGroup>
<VisualStateGroup x:Name="ActiveConectionStates">
<VisualState x:Name="NormalActiveConnectionState" />
<VisualState x:Name="ActiveConnectionInsideShape">
<Storyboard>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="ActiveSelectedBorder"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="ConnectorsAdornerVisibilityStates">
<VisualState x:Name="ConnectorsAdornerCollapsed" />
<VisualState x:Name="ConnectorsAdornerVisible">
<Storyboard>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="ConnectorsControl"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
```

```
</VisualStateGroup>
<VisualStateGroup x:Name="SelectionStates">
<VisualState x:Name="Selected" />
<VisualState x:Name="SelectedInGroup">
<Storyboard>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="SelectedBorder"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Unselected" />
<VisualState x:Name="SelectedAsGroup" />
</VisualStateGroup>
<VisualStateGroup x:Name="EditMode">
<VisualState x:Name="NormalMode" />
<VisualState x:Name="NormalEditMode">
<Storyboard>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="NormalContent"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Collapsed</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Duration="0"
Storyboard.TargetName="EditContent"
Storyboard.TargetProperty="Visibility">
<DiscreteObjectKeyFrame KeyTime="0">
<DiscreteObjectKeyFrame.Value>
<Visibility>Visible</Visibility>
</DiscreteObjectKeyFrame.Value>
</DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
```

```
</VisualState>
<VisualState x:Name="TextBoxEditMode" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Grid>
<Border x:Name="SelectedBorder"
BorderBrush="#FFADD6FF"
BorderThickness="1"
Visibility="Collapsed" />
<Border x:Name="ActiveSelectedBorder"
BorderBrush="#7FC92931"
BorderThickness="2"
Visibility="Collapsed" />
<ContentPresenter x:Name="NormalContent"
Margin="{TemplateBinding Padding}"
HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
Content="{TemplateBinding Content}"
ContentTemplate="{TemplateBinding ContentTemplate}"
telerik:DiagramBehaviors.TextWrapping="Wrap" />
<Primitives:ConnectorsControl x:Name="ConnectorsControl"
ItemContainerStyle="{TemplateBinding ConnectorStyle}"
ItemsSource="{TemplateBinding Connectors}"
Visibility="Collapsed" />
</Grid>
<Grid x:Name="PART_RotationalPart">
<ContentPresenter x:Name="EditContent"
HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
Content="{TemplateBinding Content}"
ContentTemplate="{TemplateBinding EditTemplate}"
Visibility="Collapsed" />
</Grid>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<local:ShapeStyleSelector x:Key="styleSelector"
RowStyle="{StaticResource rowStyle}"
TableStyle="{StaticResource tableStyle}" />
```

```
<Style BasedOn="{StaticResource RadDiagramConnectionStyle}"
TargetType="telerik:RadDiagramConnection">
<Setter Property="TargetCapType" Value="Arrow2Filled" />
<Setter Property="SourceCapType" Value="Arrow5Filled" />
<Setter Property="SourceCapSize" Value="5 5" />
<Setter Property="ContentTemplate">
<Setter.Value>
<DataTemplate>
<TextBlock Text="{Binding Content}" />
</DataTemplate>
</Setter.Value>
</Setter>
<Setter Property="EditTemplate">
<Setter.Value>
<DataTemplate>
<TextBlock Text="{Binding Content}" />
</DataTemplate>
</Setter.Value>
</Setter>
</Style>

<Style BasedOn="{StaticResource RadDiagramStyle}" TargetType="local:SqlDiagram" />

<Style x:Key="RadDiagramConnectorStyle" TargetType="telerik:RadDiagramConnector" />
<Style x:Key="RadDiagramConnectionStyle" TargetType="telerik:RadDiagramConnection" />
<Style x:Key="RadGeometryButtonStyle" TargetType="telerik:RadGeometryButton" />
<Style x:Key="RadGeometryToggleButtonStyle" TargetType="telerik:RadGeometryToggleButton"
/>
<Style x:Key="RadDiagramStyle" TargetType="telerik:RadDiagram" />
<Style x:Key="RadRibbonViewStyle" TargetType="telerik:RadRibbonView" />
<Style x:Key="RadDiagramRibbonStyle" TargetType="telerik:RadDiagramRibbon" />
<Style x:Key="RadTreeViewItemStyle" TargetType="telerik:RadTreeViewItem" />
<Style x:Key="RadDiagramShapeStyle" TargetType="telerik:RadDiagramShape" />
<Style x:Key="RadDiagramContainerShapeStyle"
TargetType="telerik:RadDiagramContainerShape" />
<Style x:Key="RadListBoxItemStyle" TargetType="telerik:RadListBoxItem" />
<Style x:Key="RadPanelBarItemStyle" TargetType="telerik:RadPanelBarItem" />
<Style x:Key="RadMenuStyle" TargetType="telerik:RadMenu" />
<Style x:Key="RadRadioButtonStyle" TargetType="telerik:RadRadioButton" />
<Style x:Key="RadButtonStyle" TargetType="telerik:RadButton" />
<Style x:Key="RadGeometryDropDownButtonStyle"
```

```
TargetType="telerik:RadGeometryDropDownButton" />
<Style x:Key="RadWindowStyle" TargetType="telerik:RadWindow" />
<SolidColorBrush x:Key="AccentBrush" Color="Orange" />
<SolidColorBrush x:Key="BasicBrush" Color="DarkGray" />
</ResourceDictionary>
```

RF_Cable_Map.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager.RF_Cable_Map"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:common="clr-namespace:MerlinTestStudio_Demo_Telerik.GraphViewModels"
mc:Ignorable="d"
Height="Auto" Width="Auto" d:DesignHeight="1000" d:DesignWidth="1000">

<UserControl.Resources>
<DataTemplate x:Key="contentTemplate">
<TextBlock Text="{Binding Content}"/>
</DataTemplate>
<Style TargetType="t:RadDiagramShape">
<Setter Property="Position" Value="{Binding Position}" />
</Style>

</UserControl.Resources>

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="300"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Deafult ToolBar for Diagram-->
<t:RadDiagramRibbon Diagram="{Binding ElementName=diagram}"  Grid.ColumnSpan="2" />

<!--Diagram View-->
```

```
<!-- GraphSource="{Binding Path=DiagramObj, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" -->
<t:RadDiagram x:Name="diagram" Grid.Column="0" Grid.Row="1"
IsSnapToGridEnabled="False"
IsSnapToItemsEnabled="False"
ConnectionBridge="Bow"
ConnectionTemplate="{StaticResource contentTemplate}"
ShapeTemplate="{StaticResource contentTemplate}"

ConnectorActivationChanged="diagram_ConnectorActivationChanged"
ItemsChanged="diagram_ItemsChanged"
PositionChanged="diagram_PositionChanged"
>
<!--ShapeEditTemplate="{StaticResource FILLIN}"-->
<t:RadDiagram.Resources>
<!--custom connections collections-->
<common:CustomConnectorCollection x:Key="SecondCollection">
<t:RadDiagramConnector Offset="1 0.5" Name="Connector1"/>
<t:RadDiagramConnector Offset="0 0.5" Name="Connector2"/>
</common:CustomConnectorCollection>
</t:RadDiagram.Resources>
<t:RadDiagram.ContainerShapeStyle>
<Style TargetType="t:RadDiagramContainerShape">
<Setter Property="Position" Value="{Binding Position, Mode=TwoWay}" />
<Setter Property="UseDefaultConnectors" Value="False"/>
<Setter Property="IsManipulationAdornerVisible" Value="False"/>
<Setter Property="ContentTemplate" Value="{StaticResource contentTemplate}"/>
</Style>
</t:RadDiagram.ContainerShapeStyle>
<t:RadDiagram.ShapeStyle>
<Style TargetType="t:RadDiagramShape">
<Setter Property="Position" Value="{Binding Position}" />
<Setter Property="IsManipulationAdornerVisible" Value="False"/>
<Setter Property="UseDefaultConnectors" Value="False"/>
</Style>
</t:RadDiagram.ShapeStyle>
</t:RadDiagram>

<!--TreeView for ShapeObjects-->
<!-- ItemsSource="{Binding Items}" -->
<t:RadTreeView x:Name="tree"
Grid.Column="1" Grid.Row="1"
```

```
Width="290"
ItemsSource="{Binding ElementName=diagram, Path=GraphSource.Items, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
<t:RadTreeView.ItemTemplate>
<DataTemplate>
<TextBlock Text="{Binding Content}"/>
</DataTemplate>
</t:RadTreeView.ItemTemplate>
</t:RadTreeView>
</Grid>
</UserControl>
```

AboutBox.xaml

```
<Window x:Class="MT.TestStudio.GUI.User_Controls.AboutBox"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"
Title="About Merlin Test Studio" Height="400" Width="400" WindowState="Normal"
WindowStartupLocation="CenterScreen" ResizeMode="NoResize">
<DockPanel>
<Image DockPanel.Dock="Top" Source="../Images/Logo White Background.png" Width="200"
Height="200" />

<Grid DockPanel.Dock="Top">
<!--Create a grid to hold the label and textbox-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
```

```
</Grid.RowDefinitions>

<!--Add labels-->
<Label Foreground="Black" Grid.Row="0">Merlin Test Studio</Label>
<Label Foreground="Black" Grid.Row="1" Content="{Binding Version}" />
<Label Foreground="Black" Grid.Row="2" Content="Copyright (C) 2017-2019 Merlin Test, Inc " />
<Label Foreground="Black" Grid.Row="3" Content="All rights reserved" />
<Label Foreground="Black" Grid.Row="4" Content="Merlin Test home page:" />
<TextBlock Margin="5,0,0,0" Grid.Row="5">
<Hyperlink NavigateUri="http://www.merlintest.com/"
RequestNavigate="Hyperlink_RequestNavigate">
http://www.merlintest.com/
</Hyperlink>
</TextBlock>
<Button Grid.Row="6"  Width="70" Margin="308,-25,5,5" Name="okButton"
Click="okButton_Click">OK</Button>
</Grid>
</DockPanel>
</Window>
```

AddNewFileDialog.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Dialogs.AddNewFileDialog"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.Dialogs"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

```
<t:RadListBox x:Name="NewItemListBox" Grid.Column="0" Grid.Row="0" Grid.RowSpan="3"
MouseDoubleClick="NewItemListBox_MouseDoubleClick"
SelectedItem="{Binding Path=SelectedItem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=NewItemsCollection, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">
<t:RadListBox.ItemTemplate>
<DataTemplate>
<WrapPanel>
<Image Margin="10,10,10,10" Source="{Binding ImageData}" Height="24" />
<TextBlock Margin="10,10,10,10" Text="{Binding ItemName}" FontWeight="Normal" />
</WrapPanel>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>

<TextBlock x:Name="DescriptionTextBlock"  Grid.Column="1" Grid.Row="0"
FontSize="14" Foreground="White" Margin="5"
Text="{Binding ElementName=NewItemListBox, Path=SelectedItem.Description, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">

</TextBlock>

<StackPanel Grid.Column="1" Grid.Row="1" Margin="5" >
<TextBlock Text="File Name:" Padding="0 5 5 5" FontSize="14"/>
<TextBox x:Name="CustomeFileNameTextBox"
FontSize="14" Foreground="Black"
Text="{Binding Path=CustomFileName, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">

</TextBox>
</StackPanel>

<StackPanel Grid.Column="1" Grid.Row="2" Margin="5"  Orientation="Vertical">
<t:RadButton Content="Add" Command="{Binding AddCommand}" Click="RadButton_Click"
Margin="5" FontSize="14" Padding="5">

</t:RadButton>
<t:RadButton Content="Cancel" Click="RadButton_Click"
Margin="5" FontSize="14" Padding="5">

</t:RadButton>
```

```
</StackPanel>

</Grid>
</UserControl>
```

CloseProjectSaveDiaglog.xaml

```
<UserControl x:Class="MT.TestStudio.GUI.User_Controls.CloseProjectSaveDiaglog"
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:MT.TestStudio.GUI.ViewModels"
xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
mc:Ignorable="d"
d:DesignHeight="350" d:DesignWidth="667">
<DockPanel>

<StackPanel Orientation="Vertical" DockPanel.Dock="Top" Background="#252526">

<Label Margin="10,20,10,10" Foreground="White" FontSize="14">Save changes to the following items?</Label>

<ListView Margin="10" HorizontalAlignment="Left" Background="#252526"
VerticalAlignment="Top" Width="630" Height="200" ItemsSource="{Binding FileToSaveList}" >
<ListView.ItemContainerStyle>
<Style TargetType="{x:Type ListViewItem}">
<Setter Property="FontSize" Value="14"/>
<Setter Property="Background" Value="#252526"/>
<Setter Property="Foreground" Value="White"/>
<Style.Triggers>
<Trigger Property="IsMouseOver" Value="True">
<Setter Property="Background" Value="#3E3E40" />
</Trigger>
<Trigger Property="IsSelected" Value="True">
<Setter Property="Background" Value="#3399FF" />
</Trigger>
</Style.Triggers>
</Style>
</ListView.ItemContainerStyle>
<!--<ListView.Style>
<Style TargetType="ListView">
<Setter Property="Background" Value="White"/>
```

```
</Style>

</ListView.Style>-->

</ListView>


<WrapPanel HorizontalAlignment="Right" Margin="10,10,10,10" >
<Button IsDefault="True"  Width="100" Margin="10" Content="Yes" Command="{Binding
Path=DialogButton}" CommandParameter="Yes" />
<Button Width="100" Margin="10" Content="No" Command="{Binding Path=DialogButton}"
CommandParameter="No"/>
<Button IsCancel="True" Width="100" Margin="10">_Cancel</Button>
</WrapPanel>

</StackPanel>

</DockPanel>

</UserControl>
```

ImportPinsWindowContent.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.ImportPinsWindowContent"
x:Name="ImportPinsWindow"
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:MerlinTestStudio_Demo_Telerik.ViewModels"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>

<!--Normal ListItem Style-->
<Style x:Key="NormListBoxItem" TargetType="t:RadListBoxItem">
<Setter Property="FontSize" Value="14"/>
<Setter Property="HorizontalAlignment" Value="Center"/>
</Style>
```

```
<!--Drag ListItem Style-->
<Style x:Key="DraggableListBoxItem" TargetType="t:RadListBoxItem" BasedOn="{StaticResource
NormListBoxItem}">
<Setter Property="t:DragDropManager.AllowCapturedDrag" Value="True"/>
</Style>

<Style x:Key="SelectionListBoxHeader" TargetType="TextBlock">
<Setter Property="HorizontalAlignment" Value="Center"/>
<Setter Property="VerticalAlignment" Value="Center"/>
<Setter Property="FontSize" Value="20"/>
<Setter Property="Width" Value="Auto"/>
<Setter Property="Height" Value="Auto"/>
<Setter Property="Foreground" Value="LightGray"/>
</Style>
</UserControl.Resources>

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="30"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="40"/>
<RowDefinition Height="*"/>
<RowDefinition Height="40"/>
</Grid.RowDefinitions>

<!--Import Pins List-->
<Grid Grid.Row="0">
<TextBlock Text="Import Pins" Style="{StaticResource SelectionListBoxHeader}"/>
</Grid>
<t:RadListBox x:Name="ImportPinsList" Grid.Column="0" Margin="10" Grid.Row="1"
SelectionMode="Extended"
ItemsSource="{Binding Path=ImportPinsCollection, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemContainerStyle="{StaticResource DraggableListBoxItem}"
t:ListBoxSelectedItemsBehavior.SelectedItemsSource="{Binding Path=SelectedImportPin,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
>
<t:RadListBox.DragDropBehavior>
<t:ListBoxDragDropBehavior/>
```

```
</t:RadListBox.DragDropBehavior>
</t:RadListBox>

<StackPanel Grid.Row="1" Grid.Column="1" Width="30" HorizontalAlignment="Center"
VerticalAlignment="Center">
<t:RadButton  Height="30" Margin="0 0 0 10" Command="{Binding Path=MovePinCommand}"
CommandParameter="ToImportPins" >
<t:RadButton.Content>
<StackPanel>
<Image Source="/Resources/Images/MenuItems/SingleArrow.png" Width="30" Height="30"
VerticalAlignment="Center" HorizontalAlignment="Center" RenderTransformOrigin="0.5,0.5">
<Image.RenderTransform>
<TransformGroup>
<RotateTransform Angle="-90"/>
</TransformGroup>
</Image.RenderTransform>
</Image>
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
<t:RadButton Height="30" Command="{Binding Path=MovePinCommand}"
CommandParameter="ToAvailablePins" >
<t:RadButton.Content>
<StackPanel>
<Image Source="/Resources/Images/MenuItems/SingleArrow.png" Width="30" Height="30"
VerticalAlignment="Center" HorizontalAlignment="Center" RenderTransformOrigin="0.5,0.5">
<Image.RenderTransform>
<TransformGroup>
<RotateTransform Angle="90"/>
</TransformGroup>
</Image.RenderTransform>
</Image>
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</StackPanel>

<!--Available Pins List-->
<Grid Grid.Row="0" Grid.Column="2">
<TextBlock Text="Available Pins" Style="{StaticResource SelectionListBoxHeader}"/>
</Grid>
<t:RadListBox x:Name="AvailablePinsList" Grid.Column="2" Margin="10" Grid.Row="1"
```

```
SelectionMode="Extended"
ItemsSource="{Binding Path=AvailablePinsCollection, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemContainerStyle="{StaticResource DraggableListBoxItem}"
t:ListBoxSelectedItemsBehavior.SelectedItemsSource="{Binding Path=SelectedAvailablePin,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
>
<t:RadListBox.DragDropBehavior>
<t:ListBoxDragDropBehavior AllowReorder="False"/>
</t:RadListBox.DragDropBehavior>
</t:RadListBox>

<!--Buttons Area-->
<WrapPanel Grid.Row="2" Grid.ColumnSpan="3" HorizontalAlignment="Center">
<t:RadButton Content="Cancel" Margin="0 0 40 0" Width="140" Height="30"
Command="{Binding Path=CancelBtnCommand}">
</t:RadButton>
<t:RadButton Content="Done" Width="140" Height="30"
Command="{Binding Path=DoneBtnCommand}">
</t:RadButton>
</WrapPanel>
<t:RadButton Grid.Row="2" Grid.Column="3" HorizontalAlignment="Right"
Content="Select All" Width="100" Height="30" Margin="0 0 10 10"
Command="{Binding Path=SelectAllAvailablePinsCommand}"
CommandParameter="ImportPins">
</t:RadButton>

</Grid>
</UserControl>
```

NewProjectDiaglog.xaml

```
<UserControl x:Class="MT.TestStudio.GUI.User_Controls.NewProjectDiaglog"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:local="clr-namespace:MT.TestStudio.GUI.ViewModels"
xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
mc:Ignorable="d"
d:DesignHeight="500" d:DesignWidth="945">
<DockPanel Background="#252526">
```

```
<StackPanel Orientation="Horizontal" DockPanel.Dock="Top">

<!--The Telerik control for ListView 'ListBox' for some reason doesent support 'SelectedItem' properly. ( It's only readOnly ) So to get it to work you have to create an event and then readback the IsSelected
status....easier to just use the standard ListView control and style it a little.-->
<ListView Margin="10" HorizontalAlignment="Left" Background="#252526"
VerticalAlignment="Top" Width="550" Height="200" ItemsSource="{Binding ProjectsList}"
SelectedItem="{Binding SelectedProject}">
<ListView.ItemContainerStyle>
<Style TargetType="{x:Type ListViewItem}">
<Setter Property="IsEnabled" Value="{Binding IsProjectEnabled}" />
<Setter Property="Background" Value="#252526"/>
<Setter Property="Foreground" Value="White"/>
<Style.Triggers>
<Trigger Property="IsMouseOver" Value="True">
<Setter Property="Background" Value="#3E3E40" />
</Trigger>
<Trigger Property="IsSelected" Value="True">
<Setter Property="Background" Value="#3399FF" />
</Trigger>
</Style.Triggers>
</Style>
</ListView.ItemContainerStyle>
<ListView.ItemTemplate>
<DataTemplate>
<WrapPanel>
<Image Margin="10,10,10,10" Source="{Binding ImageData}" Height="24" />
<TextBlock Margin="10,10,10,10" Text="{Binding ProjectName}" FontWeight="Normal" />

</WrapPanel>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>

<Grid Margin="10">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="350" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
```

```
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>

<TextBlock Margin="0,0,0,10" Grid.Row="0" Grid.Column="0" Foreground="White"
FontFamily="Verdana" FontSize="12" FontWeight="Bold" TextWrapping="Wrap"
Text="Description:"/>

<TextBlock Grid.Row="1" Grid.Column="0" Foreground="White" FontFamily="Verdana"
FontSize="12" FontWeight="Normal" TextWrapping="Wrap" Text="{Binding
SelectedProject.ProjectDescription}"/>
</Grid>

</StackPanel>

<Grid Margin="10">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="110" />
<ColumnDefinition Width="540" />
<ColumnDefinition Width="250" />
<ColumnDefinition Width="110" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>

<!--Add labels-->
<Label Grid.Row="0" Grid.Column="0" FontSize="12" FontWeight="Normal" Margin="5"
Foreground="White" >Project Name:</Label>
<Label Grid.Row="1" Grid.Column="0" FontSize="12" FontWeight="Normal" Margin="5"
Foreground="White" >Location:</Label>
<Label Grid.Row="3" Grid.Column="0" FontSize="12" FontWeight="Normal" Margin="5"
Foreground="White" >Test Program:</Label>
<Label Grid.Row="4" Grid.Column="0" FontSize="12" FontWeight="Normal" Margin="5"
Foreground="White" >Target System:</Label>
```

```
<!--Add Bound TextBoxes-->
<TextBox x:Name="ProjectNameTextBox" Grid.Row="0" Grid.Column="1" FontSize="12"
FontWeight="Normal" Margin="5" Background="Transparent" Foreground="White"
Text="{Binding ProjectName, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
GotMouseCapture="ProjectNameTextBox_GotMouseCapture"
LostMouseCapture="ProjectNameTextBox_LostMouseCapture"
LostKeyboardFocus="ProjectNameTextBox_LostKeyboardFocus" />
<TextBox x:Name="ProjectFilePathTextBox" Grid.Row="1" Grid.Column="1" FontSize="12"
FontWeight="Normal" Margin="5" Background="Transparent" Foreground="White" Text="{Binding
Location}" />

<t:RadComboBox Grid.Row="4" Grid.Column="1" FontSize="12" FontWeight="Normal"
Margin="5"   ToolTip="Target system for this project."
SelectedItem="{Binding UserTargetSystem}"
ItemsSource="{Binding Path=TargetSystemOptions, Mode=OneTime}">
</t:RadComboBox>
<TextBox Grid.Row="3" Grid.Column="1" FontSize="12" FontWeight="Normal" Margin="5"
Background="Transparent" Foreground="White" Text="{Binding TestProgram}" />

<Button Grid.Row="1" Grid.Column="2" Width="90" Height="25" Margin="-100,0,0,0"
Content="_Browse" Command="{Binding Path=ProjectFolderCommand}"  />

<Label Content="{Binding Path=ErrorMessage}" Grid.Row="5" Grid.Column="1"
Foreground="Red" FontSize="14" />

<WrapPanel Grid.Row="6" Grid.Column="2" HorizontalAlignment="Right" Margin="0,15,0,0"
DockPanel.Dock="Bottom">
<Button IsDefault="True"  MinWidth="60" Margin="0,0,10,0" Content="_OK" Command="{Binding
Path=CreateProjectCommand}" />
<Button IsCancel="True" MinWidth="60">_Cancel</Button>
</WrapPanel>

</Grid>

</DockPanel>

</UserControl>
```

NewProjectDialog.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Dialogs.NewProjectDialog"
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:MerlinTestStudio_Demo_Telerik.UserControls.Dialogs"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
<Grid>

</Grid>
</UserControl>
```

RemoveSiteWindow.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.RemoveSiteWindow"
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:MerlinTestStudio_Demo_Telerik.ViewModels"
xmlns:vm="clr-namespace:MerlinTestStudio_Demo_Telerik"
mc:Ignorable="d"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
d:DesignHeight="250" d:DesignWidth="400">

<UserControl.Resources>

</UserControl.Resources>

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadComboBox x:Name="ActiveSiteOptionsComboBox" Grid.ColumnSpan="2" Grid.Row="0"
Width="300" Height="40"
ItemsSource="{Binding Path=ActiveSites, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=SelectedSite, Mode=TwoWay,
```

```
UpdateSourceTrigger=PropertyChanged}">
<t:RadComboBox.ItemTemplate>
<DataTemplate x:Name="cmbDTem">
<StackPanel Orientation="Horizontal">
<TextBlock Name="Item" Text="{Binding Path=SiteName}" Width="300"/>
</StackPanel>
</DataTemplate>
</t:RadComboBox.ItemTemplate>
</t:RadComboBox>

<t:RadButton Grid.Row="1" Grid.Column="1"
Width="125" Height="25"
Content="Proceed"
Command="{Binding Path=SiteRemovalCommand}" CommandParameter="{Binding
ElementName=ActiveSiteOptionsComboBox, Path=SelectedItem}"
Click="RadButton_Click"
/>
<!--CommandParameter="{Binding RelativeSource={RelativeSource
AncestorType=t:RadWindow,Mode=FindAncestor}}"-->

</Grid>
</UserControl>
```

CalResultsExplorer.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.PaneViews.CalResultsExplorer"
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:data="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Models"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.PaneViews"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<Style TargetType="t:RadTreeViewItem" x:Key="FolderIconStyle">
<Setter Property="DefaultImageSrc" Value="FolderClosed_16x.png" />
<Setter Property="ExpandedImageSrc" Value="FolderOpened_16x.png" />
</Style>
```

```
<Style x:Key="MenuItemContainerStyle" TargetType="t:RadMenuItem">
<Setter Property="Header" Value="{Binding Path=Text}"         />
<Setter Property="ItemsSource" Value="{Binding Path=SubItems}"/>
<Setter Property="IsEnabled" Value="{Binding Path=IsEnabled}"/>
</Style>

<DataTemplate x:Key="SubCalResult">
<WrapPanel>
<t:RadGlyph Glyph="    &#xe911;" Foreground="White" Margin="0 0 5 0"/>
<TextBlock Text="{Binding ResultName}" ToolTip="{Binding ResultName}"/>
</WrapPanel>
</DataTemplate>
<HierarchicalDataTemplate x:Key="RootCalDataFile" ItemTemplate="{StaticResource
SubCalResult}"
ItemsSource="{Binding Results}" >
<TextBlock Text="{Binding FileName}" ToolTip="{Binding FilePath}" Grid.Column="1"/>
</HierarchicalDataTemplate>
</UserControl.Resources>

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
<t:RadButton Content="Add Cal Data V5"  Grid.Row="0"
Margin="5" FontSize="14" BorderBrush="DodgerBlue"
Command="{Binding Path=MVM.ImportResultsCommand}" CommandParameter="{Binding
XPath=ResultsTree.SelectedItem}"/>

<t:GroupBox Header="Result Explorer" Grid.Column="0" Grid.Row="1" FontSize="14"
BorderBrush="#FF2D2D30" BorderThickness="2" >
<Grid >
<t:RadTreeView x:Name="ResultsTree" FontSize="14"
SelectionMode="Single"
t:AnimationManager.IsAnimationEnabled="False"
ItemsSource="{Binding Path=MVM.CalibrationResultsCollection, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ItemTemplate="{StaticResource RootCalDataFile}"
ItemDoubleClick="RadTreeView_ItemDoubleClick">
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="ItemDoubleClick" Command="{Binding
Path=MVM.ResultTreeViewItemDoubleClickedCommand}" PassEventArgsToCommand="True"/>
```

```
</t:EventToCommandBehavior.EventBindings>
<t:RadTreeView.ItemContainerStyle>
<Style TargetType="t:RadTreeViewItem">
<Setter Property="IsExpanded" Value="True"/>
</Style>
</t:RadTreeView.ItemContainerStyle>
<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="TreeViewContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="TreeViewContextMenu_Opened"
ItemClick="TreeViewContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>
</t:RadTreeView>

</Grid>
</t:GroupBox>
</Grid>
</UserControl>
```

ErrorList.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.PaneViews.ErrorList"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.PaneViews"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
<Grid>
<t:RadListBox x:Name="ErrorListBox"

ItemsSource="{Binding Path=MVM.ErrorLog}">
<t:RadListBox.ItemTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="20"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
```

```
</Grid.ColumnDefinitions>
<t:RadGlyph Grid.Column="0" Glyph="&#xe11d;" FontSize="14" HorizontalAlignment="Center"
VerticalAlignment="Center" />
<TextBlock Grid.Column="1" Text="{Binding Path=ID}" FontSize="14" Foreground="White"
HorizontalAlignment="Center" VerticalAlignment="Center" />
<TextBlock Grid.Column="2" Text="{Binding Path=Name}" FontSize="14" Foreground="White"
HorizontalAlignment="Center"  VerticalAlignment="Center"/>
<!--Not correctly aligned due to the new line appended during validation result creation-->
<TextBlock Grid.Column="3" Text="{Binding Path=Description}" FontSize="14"
Foreground="White" HorizontalAlignment="Center" VerticalAlignment="Center" />
<TextBlock Grid.Column="4" Text="{Binding Path=Location}" FontSize="14" Foreground="White"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
</Grid>
</DataTemplate>
</t:RadListBox.ItemTemplate>

</t:RadListBox>
</Grid>
</UserControl>
```

Output.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.PaneViews.Output"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.PaneViews"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
<Grid Margin="0">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<StackPanel Grid.Row="0">
<t:RadButton Content="Clear All" Click="RadButton_Click"/>
</StackPanel>

<TextBox Grid.Row="1" x:Name="textBoxDebug" ScrollViewer.VerticalScrollBarVisibility="Visible"
TextWrapping="Wrap" Initialized="textBoxDebug_Initialized"
```

```
Grid.Column="2" IsEnabled="True" Background="#FF252526" Foreground="White"
FontSize="14"/>
</Grid>
</UserControl>
```

ProjectExplorer.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.PaneViews.ProjectExplorer"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:vm="clr-namespace:MerlinTestStudio_Demo_Telerik.ViewModels"
xmlns:data="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Models"
xmlns:selectors="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Selectors.StyleSelectors"
xmlns:localGUI="clr-namespace:MT.TestStudio.GUI"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.PaneViews"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<BooleanToVisibilityConverter x:Key="BoolToVis" />
<t:InvertedBooleanToVisibilityConverter x:Key="InvBoolToVis"/>

<Style TargetType="t:RadTreeViewItem" x:Key="ItemIconStyle">
<Setter Property="DefaultImageSrc" Value="{Binding Path=Icon, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" />
<!--Value="{Binding Path=Icon}"-->
</Style>
<Style TargetType="t:RadTreeViewItem" x:Key="FolderIconStyle">
<Setter Property="DefaultImageSrc" Value="FolderClosed_16x.png" />
<Setter Property="ExpandedImageSrc" Value="FolderOpened_16x.png" />
<Setter Property="IsExpanded" Value="{Binding Path=IsExpanded, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
</Style>
<Style TargetType="t:RadTreeViewItem" x:Key="ProjectIconStyle">
<Setter Property="DefaultImageSrc" Value="Merlin Icon.ico" />
<Setter Property="IsExpanded" Value="{Binding Path=IsExpanded, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
</Style>

<!--Context Menu Container Style-->
```

```
<Style x:Key="MenuItemContainerStyle" TargetType="t:RadMenuItem">
<Setter Property="Header" Value="{Binding Path=Text}" />
<Setter Property="ItemsSource" Value="{Binding Path=SubItems}"/>
<Setter Property="IsEnabled" Value="{Binding Path=IsEnabled}"/>
<Setter Property="IsSeparator" Value="{Binding Path=IsSeparator}"/>
</Style>

<DataTemplate DataType="{x:Type vm:PVM}">
<StackPanel Orientation="Horizontal" ToolTip="{Binding Path=Header, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">
<TextBlock Text="{Binding Path=Header, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />
<t:RadGlyph Margin="10 0 0 0" Glyph="&#xe109;" Visibility="{Binding Path=IsSaveRequired,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged, Converter={StaticResource
BoolToVis}}"/>
<t:RadGlyph Glyph="&#xe137;" Visibility="{Binding Path=IsHidden, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource InvBoolToVis}}" />
</StackPanel>
</DataTemplate>
<HierarchicalDataTemplate DataType="{x:Type data:ProjectFolder}" ItemsSource="{Binding
FolderItems}">
<TextBlock Text="{Binding FolderName}" />
</HierarchicalDataTemplate>
<HierarchicalDataTemplate DataType="{x:Type data:MerlinProject}" ItemsSource="{Binding
ProjectTree}">
<Grid>
<TextBlock Text="{Binding ProjectName}" />
</Grid>
</HierarchicalDataTemplate>

<selectors:ProjectTreeViewItemStyleSelector x:Key="ProjectItemStyleSelector"
DirectoryStyle="{StaticResource FolderIconStyle}"
ProjectStyle="{StaticResource ProjectIconStyle}"
FileStyle="{StaticResource ItemIconStyle}" />
</UserControl.Resources>

<Grid >
<!--IsDragDropEnabled="True"  -->
<t:RadTreeView x:Name="ProjectTree" MinWidth="220"  Margin="0,0,0,0"
IsDragDropEnabled="True"
IsDragTooltipEnabled="False"
SelectionMode="Extended"
```

```
t:AnimationManager.IsAnimationEnabled="False"

ItemDoubleClick="RadTreeView_ItemDoubleClick"
ImagesBaseDir="\Resources\Images\TreeItems\"
ItemsSource="{Binding Path=MVM.ProjectsCollection, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ItemContainerStyleSelector="{StaticResource ProjectItemStyleSelector}">
<!--ItemTemplate="{StaticResource Solution}" -->
<!--<t:RadTreeViewItem Header="{Binding Path=MVM.CurrentSolution.SolutionName,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=MVM.CurrentSolution.Projects, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ItemTemplate="{StaticResource Solution}" ItemContainerStyle="{StaticResource
ProjectIconStyle}" DefaultImageSrc="Application_16x.png"  IsExpanded="True">
</t:RadTreeViewItem>-->
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="ItemDoubleClick" Command="{Binding
Path=MVM.ProjectTreeViewItemDoubleClickedCommand}" PassEventArgsToCommand="True"/>
<t:EventBinding EventName="ItemClick" Command="{Binding
Path=MVM.ProjectTreeViewItemSingleClickedCommand}" PassEventArgsToCommand="True"/>
</t:EventToCommandBehavior.EventBindings>
<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="TreeViewContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="RadContextMenu_Opened"
ItemClick="RadContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>
</t:RadTreeView>
</Grid>
</UserControl>
```

PropertiesPane.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.PaneViews.PropertiesPane"
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:MerlinTestStudio_Demo_Telerik.UserControls.PaneViews"
xmlns:telerik="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
```

```
d:DesignHeight="450" d:DesignWidth="800">
<Grid VerticalAlignment="Top">
<Grid.RowDefinitions>
<RowDefinition Height="23" />
<RowDefinition Height="*" />
</Grid.RowDefinitions>

<Grid Height="35" VerticalAlignment="Top">
<telerik:RadButton Margin="5" HorizontalAlignment="Left" BorderThickness="0"
BorderBrush="{x:Null}">
<Grid Width="16" Height="25">
<Path Data="M8,7.6205708E-13 L10,7.6205708E-13 C10.034517,-2.980156E-08
10.068627,0.00087441405 10.102244,0.0025814176 L10.10292,0.0026024878
C11.159659,0.056168556 12,0.92994821 12,1.9999999 L14,1.9999999 L14,7 L11.73177,7
L11.710453,7.0370727 C11.359678,7.6143703 10.724874,8 10,8 L8,8 L4,8 C3.4477153,8
3,7.7761426 3,7.5 C3,7.2238574 3.4477153,7 4,7 L6.2682304,7 L6.2413893,6.9533186
C6.0874443,6.6699324 6,6.3451781 6,6 L3,6 C2.4477153,6 2,5.7761426 2,5.5 C2,5.2238574
2.4477153,5 3,5 L6,5 L6,4 L3,4 C2.4477153,4 2,3.7761424 2,3.5 C2,3.2238576 2.4477153,3 3,3
L6,3 C6,2.6548219 6.0874443,2.3300676 6.2413893,2.0466812 L6.2682304,1.9999999
L1,1.9999999 C0.44771528,2 0,1.7761424 0,1.4999999 C0,1.2238575 0.44771528,0.99999994
1,0.99999976 L7.9999914,0.99999976 L7.8977556,0.99741852 C7.3934999,0.97181356
7,0.75888348 7,0.49999997 C7,0.2238576 7.4477153,-2.980156E-08 8,7.6205708E-13 z"
Margin="2 2 0 3" Fill="DarkGray" Stretch="Fill"/>
<Path Data="M2,10 L5,10 L5,11 L2,11 z M2,8 L3,8 L3,9 L2,9 z M2,6 L3,6 L3,7 L2,7 z M0,0 L10,0
L10,1 L8,1 L8,2 L1,2 L1,12 L9,12 L9,11 L10,11 L10,13 L9,13 L0,13 L0,12 L0,2 L0,1 z"
Margin="0 0 6 0" Fill="DarkGray" Stretch="Fill" />
</Grid>
</telerik:RadButton>
<TextBlock Text="Source Control Server Properties" Margin="28 7 0 0" />
</Grid>

<StackPanel Orientation="Vertical" Grid.Row="1" Margin="5">
<telerik:RadToolBar>
<telerik:RadButton Width="24" Height="24" Margin="5">
<Path HorizontalAlignment="Center"
VerticalAlignment="Center"
Height="14"
Margin="5"
Width="14"
Stretch="Fill"
Data="M 12,13 C12,13 12,14 12,14 12,14 14,14 14,14 14,14 14,13 14,13 14,13 12,13 12,13 zM
12,8 C12,8 12,9 12,9 12,9 14,9 14,9 14,9 14,8 14,8 14,8 12,8 12,8 zM 12,6 C12,6 12,7 12,7 12,7
```

14,7 14,7 14,7 14,6 14,6 14,6 12,6 12,6 zM 12,4 C12,4 12,5 12,5 12,5 14,5 14,5 14,5 14,4 14,4 14,4 12,4 12,4 zM 12,2 C12,2 12,3 12,3 12,3 14,3 14,3 14,3 14,2 14,2 14,2 12,2 12,2 zM 6,13 C6,13 6,14 6,14 6,14 11,14 11,14 11,14 11,13 11,13 11,13 6,13 6,13 zM 6,11 C6,11 6,12 6,12 6,12 11,12 11,12 11,12 11,11 11,11 11,11 6,11 6,11 zM 6,8 C6,8 6,9 6,9 6,9 11,9 11,9 11,9 11,8 11,8 11,8 6,8 6,8 zM 6,6 C6,6 6,7 6,7 6,7 11,7 11,7 11,7 11,6 11,6 11,6 6,6 6,6 zM 6,4 C6,4 6,5 6,5 6,5 11,5 11,5 11,5 11,4 11,4 11,4 6,4 6,4 zM 6,2 C6,2 6,3 6,3 6,3 11,3 11,3 11,3 11,2 11,2 11,2 6,2 6,2 zM 6,0 C6,0 6,1 6,1 6,1 11,1 11,1 11,1 11,0 11,0 11,0 6,0 6,0 zM 2,10 C2,10 3,10 3,10 3,10 3,11 3,11 3,11 4,11 4,11 4,11 4,12 4,12 4,12 3,12 3,12 3,12 3,13 3,13 3,13 2,13 2,13 2,13 2,12 2,12 2,12 1,12 1,12 1,12 1,11 1,11 1,11 2,11 2,11 2,11 2,10 2,10 zM 0,9 C0,9 0,14 0,14 0,14 5,14 5,14 5,14 5,9 5,9 5,9 0,9 0,9 zM 2,1 C2,1 3,1 3,1 3,1 3,2 3,2 3,2 4,2 4,2 4,2 4,3 4,3 4,3 3,3 3,3 3,3 3,4 3,4 3,4 2,4 2,4 2,4 2,3 2,3 2,3 1,3 1,3 1,3 1,2 1,2 1,2 2,2 2,2 2,2 2,1 2,1 zM 0,0 C0,0 0,5 0,5 0,5 5,5 5,5 5,5 5,0 5,0 5,0 0,0 0,0 z"
Fill="Gray" />
</telerik:RadButton>
<telerik:RadButton Width="24" Height="24">
<Grid HorizontalAlignment="Center" Height="14" VerticalAlignment="Center" Width="14">
<Path Margin="8 3 0 0"
HorizontalAlignment="Left"
VerticalAlignment="Top"
Height="8"
Width="6"
Stretch="Fill"
Data="M7.4161425,5.7221889 C7.4161425,5.7221889 4,10 4,10 C4,10 0.66716588,5.7221889 0.66716588,5.7221889 C0.66716588,5.7221889 0.66716588,3.7221889 0.66716588,3.7221889 C0.66716588,3.7221889 3,6 3,6 C3,6 3,0 3,0 C3,0 5,0 5,0 C5,0 5,6 5,6 C5,6 7.4157705,3.7221889 7.4157705,3.7221889 C7.4157705,3.7221889 7.4161425,5.7221889 7.4161425,5.7221889 z"
Fill="Gray" />
<Path Data="M0,0 L7,0 L7,7 L0,7 z" Fill="DarkGray"
HorizontalAlignment="Left" Height="7" Stretch="Fill" VerticalAlignment="Top" Width="7" />
<Path Data="M2.65625,3.8643227 C2.6328125,4.0127602 2.6067708,4.1299477 2.578125,4.2158852 L1.8085938,6.4502602 L3.5234375,6.4502602 L2.7460938,4.2158852 C2.7226563,4.1429687 2.6979165,4.0257812 2.671875,3.8643227 z M2.1640625,3.0322914 L3.2109375,3.0322914 L5.2929688,8.6338539 L4.2734375,8.6338539 L3.7695313,7.2080727 L1.5664063,7.2080727 L1.0820313,8.6338539 L0.066406265,8.6338539 z"
HorizontalAlignment="Left" Height="5" RenderTransformOrigin="0.5,0.5" Stretch="Fill"
VerticalAlignment="Top" Width="5">
<Path.Fill>
<SolidColorBrush Color="White">
<SolidColorBrush.RelativeTransform>
<MatrixTransform Matrix="Identity"/>
</SolidColorBrush.RelativeTransform>

```
<SolidColorBrush.Transform>
<MatrixTransform Matrix="Identity"/>
</SolidColorBrush.Transform>
</SolidColorBrush>
</Path.Fill>
<Path.RenderTransform>
<TransformGroup>
<ScaleTransform/>
<SkewTransform/>
<RotateTransform/>
<TranslateTransform/>
</TransformGroup>
</Path.RenderTransform>
</Path>
<Path Data="M0.37109375,5.7889843 L4.5898438,5.7889843 L4.5898438,6.550703
L2.2119141,9.651289 L4.609375,9.651289 L4.609375,10.788984 L0.12207031,10.788984
L0.12207031,10.149336 L2.6318359,6.9266796 L0.37109375,6.9266796 z"
HorizontalAlignment="Left"
Height="5" Margin="1 0 0 0" RenderTransformOrigin="0.5,0.5" Stretch="Fill"
VerticalAlignment="Bottom" Width="5">
<Path.Fill>
<SolidColorBrush Color="#FF434647">
<SolidColorBrush.RelativeTransform>
<MatrixTransform Matrix="Identity"/>
</SolidColorBrush.RelativeTransform>
<SolidColorBrush.Transform>
<MatrixTransform Matrix="Identity"/>
</SolidColorBrush.Transform>
</SolidColorBrush>
</Path.Fill>
<Path.RenderTransform>
<TransformGroup>
<ScaleTransform/>
<SkewTransform/>
<RotateTransform/>
<TranslateTransform/>
</TransformGroup>
</Path.RenderTransform>
</Path>
</Grid>
</telerik:RadButton>
```

```
<telerik:RadWatermarkTextBox WatermarkContent="Search" VerticalAlignment="Center"
MinWidth="110" />
</telerik:RadToolBar>
<TextBlock Text="Property editing not available" Margin="0 10 0 0"
HorizontalAlignment="Center" />
</StackPanel>
</Grid>
</UserControl>
```

SystemExplorer.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.PaneViews.SystemExplorer"
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:MerlinTestStudio_Demo_Telerik.UserControls.PaneViews"
mc:Ignorable="d"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>

</UserControl.Resources>

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadButton Grid.Row="0" Content="Open System Config" Command="{Binding
Path=MVM.ShowSystemConfigCommand}"/>

<!--System changes under way-->
<t:RadComboBox Grid.Row="1" Grid.Column="2" FontSize="12" ToolTip="Target system for this
project." Visibility="Collapsed"
SelectedItem="{Binding Path=MVM.UserTargetSystem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=MVM.TargetSystemOptions, Mode=OneTime}">
</t:RadComboBox>
```

```
<!--Unused and Static-->
<t:RadTreeView Grid.Row="2" x:Name="SystemTree"  MinWidth="220" Margin="0,0,0,0"
Visibility="Collapsed"
SelectionMode="Single"
ImagesBaseDir="/Images/"
t:AnimationManager.IsAnimationEnabled="False" >
<t:RadTreeViewItem Header="APS-500" IsExpanded="True">
<t:RadTreeViewItem Header="Port Module" IsExpanded="True">
<t:RadTreeViewItem Header="MT-RF110"/>
<t:RadTreeViewItem Header="MT-RF210"/>
</t:RadTreeViewItem>
<t:RadTreeViewItem Header="High Power Amplifier"/>
<t:RadTreeViewItem Header="Filter Module"/>
<t:RadTreeViewItem Header="DAQmx" IsExpanded="True">
<t:RadTreeViewItem Header="Dev2"/>
<t:RadTreeViewItem Header="Dev5"/>
<t:RadTreeViewItem Header="Dev6"/>
</t:RadTreeViewItem>
<t:RadTreeViewItem Header="Digitizer"/>
<t:RadTreeViewItem Header="Signal Generators" IsExpanded="True">
<t:RadTreeViewItem Header="SC 5510A PXI"/>
</t:RadTreeViewItem>
<t:RadTreeViewItem Header="Local Oscillator"/>
<t:RadTreeViewItem Header="Power Supply" IsExpanded="True">
<t:RadTreeViewItem Header="N6700C"/>
</t:RadTreeViewItem>
<t:RadTreeViewItem Header="DownConverter" IsExpanded="True">
<t:RadTreeViewItem Header="SC 5317A PXI"/>
</t:RadTreeViewItem>
<t:RadTreeViewItem Header="Vector Signal Transceiver" IsExpanded="True">
<t:RadTreeViewItem Header="VXT PXIe vector transceiver"/>
</t:RadTreeViewItem>
<t:RadTreeViewItem Header="Reference Clock" IsExpanded="True">
<t:RadTreeViewItem Header="M9300A Reference"/>
</t:RadTreeViewItem>
<t:RadTreeViewItem Header="Unidentified" IsExpanded="True">
<t:RadTreeViewItem Header="Unknown"/>
</t:RadTreeViewItem>
</t:RadTreeViewItem>
</t:RadTreeView>
```

```
</Grid>
</UserControl>
```

DigitalLevel.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Patterns.DigitalLevel"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:conv="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Converters"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.Patterns"
mc:Ignorable="d"
d:DesignHeight="1080" d:DesignWidth="1920">

<UserControl.Resources>
<conv:NullToColorConverter x:Key="NullToColorConverter" />
</UserControl.Resources>

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<t:RadExpander Header="Digital Levels" Grid.Row="0"
VerticalContentAlignment="Top"
ExpandDirection="Down"
t:AnimationManager.IsAnimationEnabled="False"
IsExpanded="True"  >
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu Grid.Row="0">
<t:RadMenuItem Header="Insert" IsEnabled="False" Command="{Binding
Path=InsertItemCommand}" />
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Remove" IsEnabled="False" Command="{Binding
```

```
Path=RemoveItemCommand}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Duplicate" IsEnabled="False" Command="{Binding
Path=DuplicateItemCommand}"/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=CurrentLevelsData.DigiLevelsSource.Count, StringFormat={}{0}
Entries, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of time sets in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="DigitalLevelsGridView" Grid.Row="1"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Visible"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

Loaded="GridView_Loaded"
CellEditEnded="GridView_CellEditEnded"
PreparingCellForEdit="GridView_PreparingCellForEdit"

RowHeight="30"
AllowDrop="False"
CanUserDeleteRows="True"
SelectionUnit="Mixed"
SelectionMode="Extended"
SelectedItem="{Binding Path=SelectedItem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=CurrentLevelsData.DigiLevelsSource, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadGridView.Columns>
<t:GridViewComboBoxColumn Header="Pin Item" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=DigiPin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
```

```
DisplayMemberPath="PinName"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:DigitalLevel}}, Path=DataContext.GetDigitalPins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
<t:GridViewComboBoxColumn.CellTemplate>
<DataTemplate>
<TextBlock Foreground="{Binding Path=DigiPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource NullToColorConverter}}" >
<TextBlock.Text>
<PriorityBinding>
<Binding Path="DigiPin.PinName" Mode="OneWay" UpdateSourceTrigger="PropertyChanged"
IsAsync="True" />
<Binding Path="PinItem" Mode="OneWay" UpdateSourceTrigger="PropertyChanged"
IsAsync="True" />
</PriorityBinding>
</TextBlock.Text>
</TextBlock>
</DataTemplate>
</t:GridViewComboBoxColumn.CellTemplate>
<t:GridViewComboBoxColumn.CellStyle>
<Style TargetType="t:GridViewCell">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=DigiPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="ToolTip" Value="Pin not found in project"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:GridViewComboBoxColumn.CellStyle>
</t:GridViewComboBoxColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VIH, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">IH</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
```

```
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VIL, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">IL</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VOH, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">OH</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VOL, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">OL</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewComboBoxColumn Header="Termination Mode" Width="Auto"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=TerminationMode, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:DigitalLevel}}, Path=DataContext.TerminationModes, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
EditorStyle="{StaticResource NullableEnumRadComboBoxStyle}">
</t:GridViewComboBoxColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VTERM, Mode=TwoWay,
```

```
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">TERM</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VCOM, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">COM</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=IOL, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
I<Run BaselineAlignment="Subscript" FontSize="10">OL</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=IOH, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
I<Run BaselineAlignment="Subscript" FontSize="10">OH</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>

<t:GridViewDataColumn Header="Comment" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Comment, Mode=TwoWay}">
</t:GridViewDataColumn>
```

```
</t:RadGridView.Columns>
</t:RadGridView>
</Grid>
</t:RadExpander>
<t:RadExpander Header="PPMU Levels" Grid.Row="1"
VerticalContentAlignment="Top"
ExpandDirection="Down"
t:AnimationManager.IsAnimationEnabled="False"
IsExpanded="True"  >
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu Grid.Row="0">
<t:RadMenuItem Header="Insert" IsEnabled="False" Command="{Binding
Path=InsertItemCommand}" />
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Remove" IsEnabled="False" Command="{Binding
Path=RemoveItemCommand}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Duplicate" IsEnabled="False" Command="{Binding
Path=DuplicateItemCommand}"/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=CurrentLevelsData.PPMULevelsSource.Count,
StringFormat={}{0} Entries, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of time sets in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="PPMULevelsGridView" Grid.Row="1"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Visible"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
```

```
EnableRowVirtualization="True"

Loaded="GridView_Loaded"
CellEditEnded="GridView_CellEditEnded"
PreparingCellForEdit="GridView_PreparingCellForEdit"

RowHeight="30"
AllowDrop="False"
CanUserDeleteRows="True"
SelectionUnit="Mixed"
SelectionMode="Extended"
SelectedItem="{Binding Path=SelectedItem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=CurrentLevelsData.PPMULevelsSource, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadGridView.Columns>
<t:GridViewComboBoxColumn Header="Pin Item" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=DigiPin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="PinName"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:DigitalLevel}}, Path=DataContext.GetDigitalPins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
<t:GridViewComboBoxColumn.CellTemplate>
<DataTemplate>
<TextBlock Foreground="{Binding Path=DigiPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource NullToColorConverter}}" >
<TextBlock.Text>
<PriorityBinding>
<Binding Path="DigiPin.PinName" Mode="OneWay" UpdateSourceTrigger="PropertyChanged"
IsAsync="True" />
<Binding Path="PinItem" Mode="OneWay" UpdateSourceTrigger="PropertyChanged"
IsAsync="True" />
</PriorityBinding>
</TextBlock.Text>
</TextBlock>
</DataTemplate>
</t:GridViewComboBoxColumn.CellTemplate>
<t:GridViewComboBoxColumn.CellStyle>
```

```
<Style TargetType="t:GridViewCell">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=DigiPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="ToolTip" Value="Pin not found in project"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:GridViewComboBoxColumn.CellStyle>
</t:GridViewComboBoxColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VF, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">F</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=IF, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
I<Run BaselineAlignment="Subscript" FontSize="10">F</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VCH, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">CH</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VCL, Mode=TwoWay,
```

```
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">CL</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100" Header="I Range"
DataMemberBinding="{Binding Path=IRange, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewDataColumn Header="Comments" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Comment, Mode=TwoWay}">
</t:GridViewDataColumn>
</t:RadGridView.Columns>
</t:RadGridView>
</Grid>
</t:RadExpander>
<t:RadExpander Header="DC Power Levels" Grid.Row="2"
VerticalContentAlignment="Top"
ExpandDirection="Down"
t:AnimationManager.IsAnimationEnabled="False"
IsExpanded="True"  >
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu Grid.Row="0">
<t:RadMenuItem Header="Insert" IsEnabled="False" Command="{Binding
Path=InsertItemCommand}" />
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Remove" IsEnabled="False" Command="{Binding
Path=RemoveItemCommand}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Duplicate" IsEnabled="False" Command="{Binding
Path=DuplicateItemCommand}"/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=CurrentLevelsData.DCPowerLevelsSource.Count,
```

```
StringFormat={}{0} Entries, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of time sets in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="DCPowerLevelsGridView" Grid.Row="1"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Visible"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

Loaded="GridView_Loaded"
CellEditEnded="GridView_CellEditEnded"
PreparingCellForEdit="GridView_PreparingCellForEdit"

RowHeight="30"
AllowDrop="False"
CanUserDeleteRows="True"
SelectionUnit="Mixed"
SelectionMode="Extended"
SelectedItem="{Binding Path=SelectedItem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=CurrentLevelsData.DCPowerLevelsSource, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadGridView.Columns>
<t:GridViewComboBoxColumn Header="Pin Item" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=DigiPin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="PinName"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:DigitalLevel}}, Path=DataContext.GetDigitalPins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
```

```
<t:GridViewComboBoxColumn.CellTemplate>
<DataTemplate>
<TextBlock Foreground="{Binding Path=DigiPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource NullToColorConverter}}" >
<TextBlock.Text>
<PriorityBinding>
<Binding Path="DigiPin.PinName" Mode="OneWay" UpdateSourceTrigger="PropertyChanged"
IsAsync="True" />
<Binding Path="PinItem" Mode="OneWay" UpdateSourceTrigger="PropertyChanged"
IsAsync="True" />
</PriorityBinding>
</TextBlock.Text>
</TextBlock>
</DataTemplate>
</t:GridViewComboBoxColumn.CellTemplate>
<t:GridViewComboBoxColumn.CellStyle>
<Style TargetType="t:GridViewCell">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=DigiPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="ToolTip" Value="Pin not found in project"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:GridViewComboBoxColumn.CellStyle>
</t:GridViewComboBoxColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VF, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">F</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=IC, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
```

```
<TextBlock HorizontalAlignment="Center">
I<Run BaselineAlignment="Subscript" FontSize="10">C</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=IF, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
I<Run BaselineAlignment="Subscript" FontSize="10">F</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100"
DataMemberBinding="{Binding Path=VC, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
<t:GridViewDataColumn.Header>
<TextBlock HorizontalAlignment="Center">
V<Run BaselineAlignment="Subscript" FontSize="10">C</Run>
</TextBlock>
</t:GridViewDataColumn.Header>
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100" Header="I Range"
DataMemberBinding="{Binding Path=IRange, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100" Header="V Range"
DataMemberBinding="{Binding Path=VRange, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewComboBoxColumn Header="Sense" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Sense, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:DigitalLevel}}, Path=DataContext.Senses, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
```

```
EditorStyle="{StaticResource NullableEnumRadComboBoxStyle}">
</t:GridViewComboBoxColumn>
<t:GridViewDataColumn Header="Comments" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Comment, Mode=TwoWay}">
</t:GridViewDataColumn>
</t:RadGridView.Columns>
</t:RadGridView>
</Grid>
</t:RadExpander>
</Grid>
</UserControl>
```

DigitalPattern.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Patterns.DigitalPattern"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.Patterns"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<DataTemplate x:Key="EmptySelectionTemplate">
<TextBlock Text="{Binding}"  Foreground="IndianRed" FontSize="14"/>
</DataTemplate>
</UserControl.Resources>

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="5"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="5"/>
<ColumnDefinition Width="1.2*"/>
</Grid.ColumnDefinitions>
```

```
<t:RadMenu Grid.Row="0" Grid.ColumnSpan="5">
<t:RadMenuItem Header="Compile" Command="{Binding Compile_To_DPAT_Command}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Mode" Command="{Binding Path=AddModeCommand}">
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe11e;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Header="Mode" Command="{Binding Path=RemoveModeCommand}" >
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe10c;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Register" Command="{Binding Path=AddRegisterCommand}" >
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe11e;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=CurrentPattern.ModeCollection.Count, StringFormat={}{0}
Entries, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of time sets in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="Time Set"/>
<t:RadComboBox x:Name="SelectTimeSetBox" MinWidth="100"
SelectedItem="{Binding Path=CurrentPattern.TimeSetUsedObj, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="Name"
ItemsSource="{Binding Path=ParentProject.AvailableTimeSets, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
MouseDoubleClick="SelectTimeSetBox_MouseDoubleClick"
EmptySelectionBoxTemplate="{StaticResource EmptySelectionTemplate}">
<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=CurrentPattern.TimeSetUsedObj, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="EmptyText" Value="{Binding Path=CurrentPattern.TimeSetUsed,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"/>
<Setter Property="ToolTip" Value="Time Set not found in project"/>
</DataTrigger>
```

```
</Style.Triggers>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>
<t:RadMenuSeparatorItem/>
<TextBlock Text="Clock (CLK):"/>
<t:RadComboBox x:Name="ClockBox" MinWidth="100"
DisplayMemberPath="PinName"
SelectedItem="{Binding Path=CurrentPattern.ClockPin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=GetDigitalPins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EmptySelectionBoxTemplate="{StaticResource EmptySelectionTemplate}">
<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=CurrentPattern.ClockPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="EmptyText" Value="{Binding Path=CurrentPattern.ClockPinItem,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"/>
<Setter Property="ToolTip" Value="Pin not found in project"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>
<TextBlock Text="Data (DATA):"/>
<t:RadComboBox x:Name="DataBox" MinWidth="100"
DisplayMemberPath="PinName"
SelectedItem="{Binding Path=CurrentPattern.DataPin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=GetDigitalPins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EmptySelectionBoxTemplate="{StaticResource EmptySelectionTemplate}">
<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=CurrentPattern.DataPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="EmptyText" Value="{Binding Path=CurrentPattern.DataPinItem,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"/>
<Setter Property="ToolTip" Value="Pin not found in project"/>
```

```
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>
<t:RadMenuSeparatorItem/>
<t:RadToggleSwitchButton x:Name="Test_Data_Switch" Margin="5"
HorizontalAlignment="Left" VerticalAlignment="Center" Command="{Binding
IsCondensedChangedCommand}"
UncheckedContent="Expanded" CheckedContent="Condensed"
IsChecked="{Binding Path=CurrentPattern.IsCondensed, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<!--Modes Grid View-->
<t:RadGridView x:Name="DigitalPatternsGridView" Grid.Row="1" Grid.Column="0"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Collapsed"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

Loaded="GridView_Loaded"
PreparingCellForEdit="GridView_PreparingCellForEdit"
CellEditEnded="DigitalPatternsGridView_CellEditEnded"

IsExpandableBinding="{Binding IsExpandable, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
RowHeight="30"
AllowDrop="False"
CanUserDeleteRows="True"
SelectionUnit="FullRow"
SelectionMode="Single"
CurrentItem="{Binding Path=SelectedItem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
```

```
ItemsSource="{Binding Path=CurrentPattern.ModeCollection, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>

<t:RadGridView.Columns>
<t:GridViewDataColumn  Width="Auto" Header="Name"
DataMemberBinding="{Binding Path=Name, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
</t:RadGridView.Columns>

<t:RadGridView.ChildTableDefinitions>
<t:GridViewTableDefinition />
</t:RadGridView.ChildTableDefinitions>
<t:RadGridView.HierarchyChildTemplate>
<DataTemplate>
<t:RadGridView ItemsSource="{Binding Path=OpcodesDetected, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Name="childGrid"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="False"
CanUserSortColumns="False"
IsFilteringAllowed="False"
GroupRenderMode="Flat"
RowIndicatorVisibility="Collapsed"
>
<t:RadGridView.Columns>
<t:GridViewDataColumn Header="Data"
DataMemberBinding="{Binding Path=Original_Input, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center"
IsReadOnly="True"/>
<t:GridViewDataColumn Header=" Vector "
DataMemberBinding="{Binding Path=VectorNumber, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Width="Auto"
```

```
TextAlignment="Center"
IsReadOnly="True"/>
<t:GridViewDataColumn DataMemberBinding="{Binding Path=Opcode, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center"
IsReadOnly="True"/>
<t:GridViewDataColumn DataMemberBinding="{Binding Path=Section, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center"
IsReadOnly="True"/>
<t:GridViewDataColumn DataMemberBinding="{Binding Path=Comment, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center"
IsReadOnly="True"/>
</t:RadGridView.Columns>
</t:RadGridView>
</DataTemplate>
</t:RadGridView.HierarchyChildTemplate>

</t:RadGridView>

<!--Vertical Splitter-->
<GridSplitter Width="5" Grid.Row="1" Grid.Column="1" HorizontalAlignment="Stretch"
Background="Transparent" />

<!--Opcode selection control-->
<Grid Grid.Row="1" Grid.Column="2">
<Grid.RowDefinitions >
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Pattern Name Title-->
<Border Grid.Row="0" BorderBrush="DodgerBlue" BorderThickness="1" >
<Grid >
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
<TextBlock  TextAlignment="Center" FontSize="14" Padding="4"
```

```
HorizontalAlignment="Center" VerticalAlignment="Center"
Text="{Binding Path=SelectedItem.Name, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">
</TextBlock>
<!--Horizontal Input Values List-->
<t:RadListBox Grid.Row="1" BorderBrush="Transparent"
SelectedIndex="{Binding Path=SelectedVector, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=SelectedVectorItem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=SelectedItem.ModeVectors, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ScrollViewer.HorizontalScrollBarVisibility="Hidden"
SelectionMode="Single">
<t:RadListBox.ItemsPanel>
<ItemsPanelTemplate>
<WrapPanel IsItemsHost="True" />
</ItemsPanelTemplate>
</t:RadListBox.ItemsPanel>
<t:RadListBox.ItemTemplate>
<DataTemplate>
<Border BorderThickness="1" BorderBrush="DodgerBlue" CornerRadius="4" MinWidth="50"  >
<TextBlock  TextAlignment="Center" FontSize="14" Margin="2"
HorizontalAlignment="Center" VerticalAlignment="Center"
Text="{Binding Path=InputValue, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}">
</TextBlock>
</Border>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>
</Grid>
</Border>

<!--Vector List Headers-->

<Border Grid.Row="3" BorderBrush="Transparent" BorderThickness="1" >
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

```
<Grid Grid.Row="0" Margin="2">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="2.5*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<TextBlock Text="Vector" Grid.Column="0" HorizontalAlignment="Center"/>
<TextBlock Text="Opcode" Grid.Column="1" HorizontalAlignment="Center"/>
<TextBlock Text="{Binding Path=CurrentPattern.ClockPin.PinName, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Grid.Column="2" HorizontalAlignment="Center"/>
<TextBlock Text="{Binding Path=CurrentPattern.DataPin.PinName, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Grid.Column="3" HorizontalAlignment="Center"/>
<TextBlock Text="Comment" Grid.Column="4"  HorizontalAlignment="Center"/>
</Grid>

<t:RadListBox Grid.Row="1"
ItemsSource="{Binding Path=SelectedVectorizedData, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
SelectionMode="Single" BorderBrush="Transparent">
<!--Vector opcode editor items with Operation object data templete for selecting Opcodes-->
<t:RadListBox.ItemTemplate>
<DataTemplate>
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="3*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<TextBlock Grid.Column="0" Text="{Binding Path=VectorNumber}" HorizontalAlignment="Center"
/>

<t:RadComboBox Grid.Column="1"

SelectedValue="{Binding Path=Opcode}"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:DigitalPattern}}, Path=DataContext.Opcodes, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
```

```
EmptyText="-"
ClearSelectionButtonVisibility="Visible"
ClearSelectionButtonContent="Clear"
>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type local:DigitalPattern}},
Path=DataContext.OpcodeSelectedCommand}" RaiseOnHandledEvents="False" />
</t:EventToCommandBehavior.EventBindings>
</t:RadComboBox>

<TextBlock Grid.Column="2" Text="{Binding Path=Clock}" HorizontalAlignment="Center" />
<TextBlock Grid.Column="3" Text="{Binding Path=Data}"  HorizontalAlignment="Center"/>
<TextBlock Grid.Column="4" Text="{Binding Path=Comment}"  HorizontalAlignment="Center"/>
</Grid>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>
</Grid>
</Border>

</Grid>

<!--Vertical Splitter-->
<GridSplitter Width="5" Grid.Row="1" Grid.Column="3" HorizontalAlignment="Stretch"
Background="Transparent" />

<!--DPATSRC File Live View-->
<t:RadSyntaxEditor x:Name="syntaxEditor" Grid.Row="1" Grid.Column="4">

</t:RadSyntaxEditor>
</Grid>
</UserControl>
```

DigitalTiming.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Patterns.DigitalTiming"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:conv="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Converters"
```

```
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.Patterns"
mc:Ignorable="d"
d:DesignHeight="1080" d:DesignWidth="1920">

<UserControl.Resources>
<conv:NullToColorConverter x:Key="NullToColorConverter" />
</UserControl.Resources>

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu Grid.Row="0">
<t:RadMenuItem Header="Insert" Command="{Binding Path=InsertItemCommand}" />
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Remove" Command="{Binding Path=RemoveItemCommand}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Duplicate" Command="{Binding Path=DuplicateItemCommand}"/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=CurrentTimingData.DigiTimeSource.Count, StringFormat={}{0}
Entries, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of time sets in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="DigiTimingGridView" Grid.Row="1"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Visible"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

Loaded="GridView_Loaded"
PreparingCellForEdit="GridView_PreparingCellForEdit"
```

```
CellEditEnded="DigiTimingGridView_CellEditEnded"

RowHeight="30"
AllowDrop="False"
CanUserDeleteRows="True"
SelectionUnit="Mixed"
SelectionMode="Extended"
SelectedItem="{Binding Path=SelectedItem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=CurrentTimingData.DigiTimeSource, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadGridView.Columns>
<t:GridViewDataColumn  Width="100" Header="Name"
DataMemberBinding="{Binding Path=Name, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100" Header="Period (ns)"
DataMemberBinding="{Binding Path=Period, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewComboBoxColumn Header="Pin Item" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=DigiPin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayMemberPath="PinName"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:DigitalTiming}}, Path=DataContext.GetDigitalPins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
<t:GridViewComboBoxColumn.CellTemplate>
<DataTemplate>
<TextBlock Foreground="{Binding Path=DigiPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged, Converter={StaticResource NullToColorConverter}}" >
<TextBlock.Text>
<PriorityBinding>
<Binding Path="DigiPin.PinName" Mode="OneWay" UpdateSourceTrigger="PropertyChanged"
IsAsync="True" />
<Binding Path="PinItem" Mode="OneWay" UpdateSourceTrigger="PropertyChanged"
IsAsync="True" />
```

```
</PriorityBinding>
</TextBlock.Text>
</TextBlock>
</DataTemplate>
</t:GridViewComboBoxColumn.CellTemplate>
<t:GridViewComboBoxColumn.CellStyle>
<Style TargetType="t:GridViewCell">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=DigiPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="ToolTip" Value="Pin not found in project"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:GridViewComboBoxColumn.CellStyle>
</t:GridViewComboBoxColumn>
<t:GridViewComboBoxColumn Header="Edge Multiplier" Width="Auto"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=EdgeMultiplier, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:DigitalTiming}}, Path=DataContext.EdgeMultipliers, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
</t:GridViewComboBoxColumn>
<t:GridViewComboBoxColumn Header="Drive Format" Width="Auto"
HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=DriveFormat, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:DigitalTiming}}, Path=DataContext.DriveFormats, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
</t:GridViewComboBoxColumn>
<t:GridViewDataColumn  Width="100" Header="Drive On"
DataMemberBinding="{Binding Path=DriveOn, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100" Header="Drive Data (ns)"
```

```
DataMemberBinding="{Binding Path=DriveData, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100" Header="Drive Return (ns)"
DataMemberBinding="{Binding Path=DriveReturn, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100" Header="Drive Off (ns)"
DataMemberBinding="{Binding Path=DriveOff, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewDataColumn  Width="100" Header="Compare Strobe (ns)"
DataMemberBinding="{Binding Path=CompareStrobe, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center">
</t:GridViewDataColumn>
<t:GridViewDataColumn Header="Comment" Width="Auto" HeaderTextAlignment="Center"
DataMemberBinding="{Binding Path=Comment, Mode=TwoWay}">
</t:GridViewDataColumn>
</t:RadGridView.Columns>
</t:RadGridView>



</Grid>
</UserControl>
```

GenericRffePattern.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Patterns.GenericRffePattern"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:eSourceExt="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
xmlns:e="clr-namespace:MerlinTestStudio_Demo_Telerik.Data"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.Patterns"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
```

```
<UserControl.Resources>

<DataTemplate x:Key="EmptySelectionTemplate">
<TextBlock Text="{Binding}"  Foreground="IndianRed" FontSize="14"/>
</DataTemplate>

<!--Disabled Cell styles-->
<Style x:Key="MaskedWriteGridViewCellStyle" TargetType="t:GridViewCell">
<Setter Property="IsEnabled" Value="False"></Setter>
<Setter Property="Foreground" Value="IndianRed"></Setter>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=Spec, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="3">
<Setter Property="IsEnabled" Value="True"></Setter>
<Setter Property="Foreground" Value="White"></Setter>
</DataTrigger>
</Style.Triggers>
</Style>
<Style x:Key="RegisterZeroWriteGridViewCellStyle" TargetType="t:GridViewCell">
<Setter Property="IsEnabled" Value="True"></Setter>
<Setter Property="Foreground" Value="White"></Setter>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=Spec, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="2">
<Setter Property="IsEnabled" Value="False"></Setter>
<Setter Property="Foreground" Value="IndianRed"></Setter>
</DataTrigger>
</Style.Triggers>
</Style>
<Style x:Key="ByteCountGridViewCellStyle" TargetType="t:GridViewCell">
<Setter Property="IsEnabled" Value="False"></Setter>
<Setter Property="Foreground" Value="IndianRed"></Setter>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=Spec, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="4">
<Setter Property="IsEnabled" Value="True"></Setter>
<Setter Property="Foreground" Value="White"></Setter>
</DataTrigger>
<DataTrigger Binding="{Binding Path=Spec, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="5">
<Setter Property="IsEnabled" Value="True"></Setter>
```

```
<Setter Property="Foreground" Value="White"></Setter>
</DataTrigger>
<DataTrigger Binding="{Binding Path=Spec, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="6">
<Setter Property="IsEnabled" Value="True"></Setter>
<Setter Property="Foreground" Value="White"></Setter>
</DataTrigger>
<DataTrigger Binding="{Binding Path=Spec, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="7">
<Setter Property="IsEnabled" Value="True"></Setter>
<Setter Property="Foreground" Value="White"></Setter>
</DataTrigger>
</Style.Triggers>
</Style>

<!--Invalid/Valid Row Style-->
<Style TargetType="t:GridViewRow" x:Key="CustomErrorGridViewRowStyle">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=Status, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="2">
<Setter Property="BorderBrush" Value="Red"/>
<Setter Property="BorderThickness" Value="1"/>
<Setter Property="Background" Value="#40282d"/>
<Setter Property="SelectedBackground" Value="#cf4e4e"/>
<Setter Property="ToolTip" Value="{Binding Path=ErrorMessage, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"/>
</DataTrigger>
</Style.Triggers>
</Style>
</UserControl.Resources>

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="5"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="5"/>
<ColumnDefinition >
```

```
<ColumnDefinition.Style>
<Style TargetType="ColumnDefinition">
<Setter Property="Width" Value="1.2*"/>
<Style.Triggers>
<DataTrigger Binding="{Binding ElementName=Preview_Visibility_Switch, Path=IsChecked,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}" Value="False">
<Setter Property="Width" Value="0"/>
</DataTrigger>
</Style.Triggers>
</Style>
</ColumnDefinition.Style>
</ColumnDefinition>
</Grid.ColumnDefinitions>

<t:RadMenu Grid.Row="0" Grid.ColumnSpan="5">
<t:RadMenuItem Header="Compile" Command="{Binding Compile_To_DPAT_Command}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Mode" Command="{Binding Path=AddModeCommand}">
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe11e;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Header="Mode" Command="{Binding Path=RemoveModeCommand}" >
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe10c;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Data to All" Command="{Binding Path=AddRegisterCommand}" >
<t:RadMenuItem.Icon >
<t:RadGlyph Glyph="&#xe11e;" Foreground="White" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=CurrentPattern.ModeCollection.Count, StringFormat={}{0}
Entries, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of time sets in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="Time Set"/>
<t:RadComboBox x:Name="SelectTimeSetBox" MinWidth="100"
SelectedItem="{Binding Path=CurrentPattern.TimeSetUsedObj, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
```

```
DisplayMemberPath="Name"
ItemsSource="{Binding Path=ParentProject.AvailableTimeSets, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
MouseDoubleClick="SelectTimeSetBox_MouseDoubleClick"
EmptySelectionBoxTemplate="{StaticResource EmptySelectionTemplate}">
<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=CurrentPattern.TimeSetUsedObj, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="EmptyText" Value="{Binding Path=CurrentPattern.TimeSetUsed,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"/>
<Setter Property="ToolTip" Value="Time Set not found in project"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>
<t:RadMenuSeparatorItem/>
<TextBlock Text="Clock (CLK):"/>
<t:RadComboBox x:Name="ClockBox" MinWidth="100"
DisplayMemberPath="PinName"
SelectedItem="{Binding Path=CurrentPattern.ClockPin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=GetDigitalPins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EmptySelectionBoxTemplate="{StaticResource EmptySelectionTemplate}">
<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=CurrentPattern.ClockPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="EmptyText" Value="{Binding Path=CurrentPattern.ClockPinItem,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"/>
<Setter Property="ToolTip" Value="Pin not found in project"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>
<TextBlock Text="Data (DATA):"/>
<t:RadComboBox x:Name="DataBox" MinWidth="100"
```

```
DisplayMemberPath="PinName"
SelectedItem="{Binding Path=CurrentPattern.DataPin, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=GetDigitalPins, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EmptySelectionBoxTemplate="{StaticResource EmptySelectionTemplate}">
<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Style.Triggers>
<DataTrigger Binding="{Binding Path=CurrentPattern.DataPin, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="{x:Null}" >
<Setter Property="EmptyText" Value="{Binding Path=CurrentPattern.DataPinItem,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"/>
<Setter Property="ToolTip" Value="Pin not found in project"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>
<t:RadMenuSeparatorItem/>
<t:RadToggleSwitchButton x:Name="Test_Data_Switch" Margin="5"
HorizontalAlignment="Left" VerticalAlignment="Center" Command="{Binding
IsCondensedChangedCommand}"
UncheckedContent="Expanded" CheckedContent="Condensed"
IsChecked="{Binding Path=CurrentPattern.IsCondensed, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"/>
<t:RadMenuSeparatorItem/>
<t:RadToggleSwitchButton x:Name="Preview_Visibility_Switch" Margin="5"
HorizontalAlignment="Left" VerticalAlignment="Center"
UncheckedContent="Open Preview" CheckedContent="Close Preview"
IsChecked="True"/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<!--Modes Grid View-->
<t:RadGridView x:Name="DigitalPatternsGridView" Grid.Row="1" Grid.Column="0"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"
```

```
RowIndicatorVisibility="Collapsed"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

Loaded="GridView_Loaded"
PreparingCellForEdit="GridView_PreparingCellForEdit"
CellEditEnded="DigitalPatternsGridView_CellEditEnded"

IsExpandableBinding="{Binding IsExpandable, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
IsExpandedBinding="{Binding IsExpanded, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
RowHeight="30"
RowStyle="{StaticResource CustomErrorGridViewRowStyle}"
AllowDrop="False"
CanUserDeleteRows="True"
SelectionUnit="FullRow"
SelectionMode="Single"
CurrentItem="{Binding Path=SelectedItem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=CurrentPattern.ModeCollection, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>


<t:RadGridView.HierarchyChildTemplate>
<DataTemplate>
<t:RadGridView ItemsSource="{Binding Path=ModeVectors, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Name="childGrid"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="False"
CanUserSortColumns="False"
IsFilteringAllowed="False"
```

```
GroupRenderMode="Flat"
RowIndicatorVisibility="Collapsed"
RowStyle="{StaticResource CustomErrorGridViewRowStyle}"
CellEditEnded="ChildGridView_CellEditEnded"
SelectedItem="{Binding Path=DataContext.ChildGridSelectedGenericModeVector,
RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type local:GenericRffePattern}},
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
>
<t:RadGridView.Columns>
<t:GridViewComboBoxColumn Header="Spec"
DataMemberBinding="{Binding Path=Spec, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSourceBinding="{Binding Source={eSourceExt:EnumBindingSource {x:Type
e:RffePatternSpec}}}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}">
</t:GridViewComboBoxColumn>

<t:GridViewDataColumn Header="User ID"
DataMemberBinding="{Binding Path=UserIDValue, StringFormat='0x{0}',  Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center"/>

<t:GridViewDataColumn Header="Byte Count"
DataMemberBinding="{Binding Path=ByteCountBinary,  Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center" CellStyle="{StaticResource ByteCountGridViewCellStyle}" />

<t:GridViewDataColumn Header="Register"
DataMemberBinding="{Binding Path=RegisterValue, StringFormat='0x{0}', Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Width="Auto"
TextAlignment="Center" CellStyle="{StaticResource RegisterZeroWriteGridViewCellStyle}">
</t:GridViewDataColumn>

<t:GridViewDataColumn Header="Mask"
DataMemberBinding="{Binding Path=MaskValue, StringFormat='0x{0}', Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Width="Auto"
TextAlignment="Center" CellStyle="{StaticResource MaskedWriteGridViewCellStyle}">
</t:GridViewDataColumn>

<t:GridViewDataColumn Header="Data"
DataMemberBinding="{Binding Path=InputValue, StringFormat='0x{0}', Mode=OneWay,
```

```
UpdateSourceTrigger=PropertyChanged}" Width="Auto"
TextAlignment="Center"/>
</t:RadGridView.Columns>
</t:RadGridView>
</DataTemplate>
</t:RadGridView.HierarchyChildTemplate>

<t:RadGridView.Columns>
<t:GridViewDataColumn Header="Pattern Name" TextAlignment="Center"
DataMemberBinding="{Binding Path=Name, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
>
</t:GridViewDataColumn>
<t:GridViewDataColumn Header="Entry Count" TextAlignment="Center"
DataMemberBinding="{Binding Path=ModeVectors.Count, StringFormat={}{0} Entries,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}">
</t:GridViewDataColumn>
</t:RadGridView.Columns>

</t:RadGridView>

<!--Vertical Splitter-->
<GridSplitter Width="5" Grid.Row="1" Grid.Column="1" HorizontalAlignment="Stretch"
Background="Transparent" />

<!--Opcode selection control-->
<Grid Grid.Row="1" Grid.Column="2">
<Grid.RowDefinitions >
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Pattern Name Title-->
<Border Grid.Row="0" BorderBrush="DodgerBlue" BorderThickness="1" >
<Grid >
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
<TextBlock  TextAlignment="Center" FontSize="14" Padding="4"
```

```
HorizontalAlignment="Center" VerticalAlignment="Center"
Text="{Binding Path=SelectedItem.Name, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">
</TextBlock>
<!--Horizontal Input Values List-->
<t:RadListBox Grid.Row="1" BorderBrush="Transparent"
SelectedIndex="{Binding Path=SelectedVector, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=SelectedVectorItem, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=SelectedItem.ModeVectors, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ScrollViewer.HorizontalScrollBarVisibility="Visible"
SelectionMode="Single">
<t:RadListBox.ItemsPanel>
<ItemsPanelTemplate>
<WrapPanel IsItemsHost="True" />
</ItemsPanelTemplate>
</t:RadListBox.ItemsPanel>
<t:RadListBox.ItemTemplate>
<DataTemplate>
<Border BorderThickness="1" BorderBrush="DodgerBlue" CornerRadius="4" MinWidth="50"  >
<TextBlock  TextAlignment="Center" FontSize="14" Margin="2"
HorizontalAlignment="Center" VerticalAlignment="Center"
Text="{Binding Path=InputValue, StringFormat='0x{0}', Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">
</TextBlock>
</Border>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>
</Grid>
</Border>

<!--Vector List Headers-->

<Border Grid.Row="3" BorderBrush="Transparent" BorderThickness="1" >
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

```
<Grid Grid.Row="0" Margin="2">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="2.5*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<TextBlock Text="Vector" Grid.Column="0" HorizontalAlignment="Center"/>
<TextBlock Text="Opcode" Grid.Column="1" HorizontalAlignment="Center"/>
<TextBlock Text="{Binding Path=CurrentPattern.ClockPin.PinName, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Grid.Column="2" HorizontalAlignment="Center"/>
<TextBlock Text="{Binding Path=CurrentPattern.DataPin.PinName, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Grid.Column="3" HorizontalAlignment="Center"/>
<TextBlock Text="Comment" Grid.Column="4"  HorizontalAlignment="Center"/>
</Grid>

<t:RadListBox Grid.Row="1"
ItemsSource="{Binding Path=SelectedVectorizedData, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
SelectionMode="Single" BorderBrush="Transparent">
<!--Vector opcode editor items with Operation object data templete for selecting Opcodes-->
<t:RadListBox.ItemTemplate>
<DataTemplate>
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="3*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<TextBlock Grid.Column="0" Text="{Binding Path=VectorNumber}" HorizontalAlignment="Center"
/>

<t:RadComboBox Grid.Column="1"

SelectedValue="{Binding Path=Opcode}"
ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:GenericRffePattern}}, Path=DataContext.Opcodes, Mode=OneWay,
```

```
UpdateSourceTrigger=PropertyChanged}"
EmptyText="-"
ClearSelectionButtonVisibility="Visible"
ClearSelectionButtonContent="Clear"
>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type local:GenericRffePattern}},
Path=DataContext.OpcodeSelectedCommand}" RaiseOnHandledEvents="False" />
</t:EventToCommandBehavior.EventBindings>
</t:RadComboBox>

<TextBlock Grid.Column="2" Text="{Binding Path=Clock}" HorizontalAlignment="Center" />
<TextBlock Grid.Column="3" Text="{Binding Path=Data}"  HorizontalAlignment="Center"/>
<TextBlock Grid.Column="4" Text="{Binding Path=Comment}"  HorizontalAlignment="Center"/>
</Grid>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>
</Grid>
</Border>

</Grid>

<!--Vertical Splitter-->
<GridSplitter Width="5" Grid.Row="1" Grid.Column="3" HorizontalAlignment="Stretch"
Background="Transparent" />

<!--DPATSRC File Live View-->
<t:RadSyntaxEditor x:Name="syntaxEditor" Grid.Row="1" Grid.Column="4" >
<t:RadSyntaxEditor.Style>
<Style TargetType="t:RadSyntaxEditor">
<Setter Property="Visibility" Value="Visible"/>
<Style.Triggers>
<DataTrigger Binding="{Binding ElementName=Preview_Visibility_Switch, Path=IsChecked,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}" Value="False">
<Setter Property="Visibility" Value="Collapsed"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadSyntaxEditor.Style>
</t:RadSyntaxEditor>
```

```
</Grid>
</UserControl>
```

ConditionsData.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestConditions.ConditionsData"
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:MerlinTestStudio_Demo_Telerik.ViewModels"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:ss="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Selectors.StyleSelectors"
mc:Ignorable="d"
Height="Auto"  Width="Auto" Unloaded="UserControl_Unloaded">

<UserControl.Resources>
<Style x:Key="MenuItemContainerStyle" TargetType="t:RadMenuItem">
<Setter Property="Header" Value="{Binding Path=Text}"          />
<Setter Property="ItemsSource" Value="{Binding Path=SubItems}"/>
<Setter Property="IsEnabled" Value="{Binding Path=IsEnabled}"/>
<Setter Property="Icon">
<Setter.Value>
<Image Source="{Binding Path=IconPath}" Width="20" Height="20"/>
</Setter.Value>
</Setter>
</Style>

<!--Flipped Image Style-->
<Style TargetType="Image" x:Key="FlippedImage">
<Setter Property="RenderTransformOrigin" Value="0.5,0.5"/>
<Setter Property="RenderTransform">
<Setter.Value>
<TransformGroup>
<RotateTransform Angle="180"/>
</TransformGroup>
</Setter.Value>
</Setter>
</Style>

</UserControl.Resources>
```

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<Grid Grid.Row="0">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<!--Toolbar-->
<t:RadMenu Grid.Column="0" >
<t:RadMenuItem Header="Product Info" x:Name="ProductInfoBtn" Command="{Binding
Path=OpenProductInfoCommand}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Add Test" Command="{Binding Path=AddTestCommand}" />
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Add Parameter" x:Name="AddTestParameterBtn" Command="{Binding
Path=AddTestParameterCommand}"/>
<t:RadMenuSeparatorItem/>

<!-- Renumerate Tests MenuItem-->
<t:RadMenuItem Header="Renumerate" Command="{Binding Path=RenumerateTestsCommand}"
/>
<t:RadMenuSeparatorItem/>

<!-- UpDownByOne MenuItem(s)-->
<t:RadMenuItem  Padding="2" ToolTip="Moves the selected row up by 1" Command="{Binding
Path=UpOrDownCommand}" CommandParameter="Up">
<t:RadMenuItem.Header>
<Image Source="/Resources/Images/MenuItems/SingleArrow.png" Width="30" Height="30"/>
</t:RadMenuItem.Header>
</t:RadMenuItem>
<t:RadMenuItem Padding="2" ToolTip="Moves the selected row down by 1" Command="{Binding
Path=UpOrDownCommand}" CommandParameter="Down">
<t:RadMenuItem.Header>
<Image Source="/Resources/Images/MenuItems/SingleArrow.png" Width="30" Height="30"
Style="{StaticResource FlippedImage}"/>
</t:RadMenuItem.Header>
```

```
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>

<!-- Test SubMenu (Collapsed) -->
<t:RadMenuItem Header="Test" Visibility="Collapsed" IsEnabled="False" >
<!-- Insert Test MenuItem-->
<t:RadMenuItem Header="Insert New" Command="{Binding Path=InsertTestCommand}">
<t:RadMenuItem.Icon>
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="20" Height="20"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<!-- Remove Test MenuItem-->
<t:RadMenuItem Header="Remove" Command="{Binding Path=RemoveTestCommand}" >
<t:RadMenuItem.Icon>
<Image Source="/Resources/Images/MenuItems/RemoveIcon.png" Width="20" Height="20" />
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<!-- Duplicate Test MenuItem-->
<t:RadMenuItem Header="Duplicate" Command="{Binding Path=DuplicateTestCommand}" >
<t:RadMenuItem.Icon>
<Image Source="/Resources/Images/MenuItems/Clone.png" Width="20" Height="20"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
</t:RadMenuItem>

<TextBlock Text="{Binding Path=TestCollection.Count, StringFormat={}{0} Entries,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of tests in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<!-- MoveToTool MenuItem-->
<StackPanel Orientation="Horizontal" Grid.Column="1" HorizontalAlignment="Left"
VerticalAlignment="Center" Margin="10 0 10 0">
<TextBlock Text="Place Selected to Test # :" VerticalAlignment="Center" Margin="0 0 8 0"/>
<t:RadNumericUpDown x:Name="LocationInput" Width="40" Padding="4"
KeyUp="LocationInput_KeyUp"
NumberDecimalDigits="0"
Minimum="1"
Maximum="{Binding ElementName=ConditionsGrid, Path=ItemsSource.Count}">
```

```
<t:RadNumericUpDown.InputBindings>
<KeyBinding Command="{Binding Path=MoveSelectedToCommand}"
CommandParameter="{Binding ElementName=LocationInput, Path=Value}" Key="Enter" />
</t:RadNumericUpDown.InputBindings>
</t:RadNumericUpDown>
</StackPanel>
</Grid>

<!--GridView-->
<t:RadGridView   Grid.Row="1"
Margin="5"
x:Name="ConditionsGrid"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
GroupRenderMode="Flat"
CanUserSortColumns="False"
CanUserDeleteRows="False"
IsFilteringAllowed="False"
RowIndicatorVisibility="Collapsed"
ShowGroupPanel="False"
AllowDrop="True"

SelectionMode="Single"
SelectionUnit="FullRow"

DataLoadMode="Asynchronous"
EnableRowVirtualization="True"
EnableColumnVirtualization="True"

SelectedItem="{Binding Path=SelectedTest, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=TestCollection, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
Loaded="ConditionsGrid_Loaded"
CellEditEnded="ConditionsGrid_CellEditEnded"
>

<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="CellEditEnded" Command="{Binding Path=CellEditedCommand}"
RaiseOnHandledEvents="True"/>
</t:EventToCommandBehavior.EventBindings>
<t:RadContextMenu.ContextMenu>
```

```
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>
<t:RadGridView.RowStyleSelector>
<ss:TestStyleSelector/>
</t:RadGridView.RowStyleSelector>
<t:RadGridView.Resources>
<DataTemplate x:Key="DraggedItemTemplate">
<StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="Dragging:" />
<TextBlock Text="{Binding CurrentDraggedItem}"
FontWeight="Bold" />
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="{Binding CurrentDropPosition}"
FontWeight="Bold"
MinWidth="45" />
<TextBlock Text=", ("
Foreground="Gray" />
<TextBlock Text="{Binding CurrentDraggedOverItem}" />
<TextBlock Text=")"
Foreground="Gray" />
</StackPanel>
</StackPanel>
</DataTemplate>
</t:RadGridView.Resources>

</t:RadGridView>


</Grid>
</UserControl>
```

GoldLimits.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestLimits.GoldLimits"
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:eSourceExt="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
```

```
xmlns:e="clr-namespace:MerlinTestStudio.DataModels;assembly=MerlinTestStudio.DataModels"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.TestLimits"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
<UserControl.Resources>
</UserControl.Resources>

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu>
<t:RadMenuItem Header="Add Test" IsEnabled="False"/>
<t:RadMenuItem Header="Remove Test"  IsEnabled="False"/>
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=GoldLimitsObj.Data.Count, StringFormat={}{0} Entries,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of tests in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="GoldLimitsGridView" Grid.Row="1"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Visible"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

Loaded="GoldLimitsGridView_Loaded"
CellEditEnded="GoldLimitsGridView_CellEditEnded"

RowHeight="30"
AllowDrop="False"
```

```
CanUserDeleteRows="True"
SelectedItem="{Binding Path=SelectedLimit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=GoldLimitsObj.Data, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>

<t:RadGridView.ColumnGroups>
<t:GridViewColumnGroup Name="TestInfo" Header="TEST INFO" >
<t:GridViewColumnGroup.HeaderTemplate>
<DataTemplate>
<TextBlock Text="TEST INFO" VerticalAlignment="Center" HorizontalAlignment="Center"/>
</DataTemplate>
</t:GridViewColumnGroup.HeaderTemplate>
</t:GridViewColumnGroup>
<t:GridViewColumnGroup Name="Gold" Header="GOLDS" >
<t:GridViewColumnGroup.HeaderTemplate>
<DataTemplate>
<TextBlock Text="GOLDS" VerticalAlignment="Center" HorizontalAlignment="Center"/>
</DataTemplate>
</t:GridViewColumnGroup.HeaderTemplate>
</t:GridViewColumnGroup>
</t:RadGridView.ColumnGroups>

<t:RadGridView.Columns>
<t:GridViewDataColumn Header="Test Number"  TextAlignment="Center" IsVisible="True"
DataMemberBinding="{Binding Path=TestNumber, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"  ColumnGroupName="TestInfo"/>
<t:GridViewDataColumn Header="Test Name"  HeaderTextAlignment="Center"
TextAlignment="Left" IsVisible="True"
DataMemberBinding="{Binding Path=TestName, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" ColumnGroupName="TestInfo"/>
<t:GridViewDataColumn Header="Units"  HeaderTextAlignment="Center" TextAlignment="Center"
IsVisible="True"
DataMemberBinding="{Binding Path=Units, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" ColumnGroupName="TestInfo"/>
```

```
<!--Golds-->
<t:GridViewComboBoxColumn Header="Retest Control"  HeaderTextAlignment="Center"
TextAlignment="Center" Style="{StaticResource TestLimits_ColumnStyle}"
DataMemberBinding="{Binding Path=GoldRetestControl, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSourceBinding="{Binding Source={eSourceExt:EnumBindingSource {x:Type
e:GoldControl}}}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}"
ColumnGroupName="Gold">
</t:GridViewComboBoxColumn>
<t:GridViewDataColumn Header="Retest Lower" HeaderTextAlignment="Center"
TextAlignment="Center" Style="{StaticResource TestLimits_ColumnStyle}"
DataMemberBinding="{Binding Path=GoldRetestLower, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" ColumnGroupName="Gold"/>
<t:GridViewDataColumn Header="Retest Upper" HeaderTextAlignment="Center"
TextAlignment="Center" Style="{StaticResource TestLimits_ColumnStyle}"
DataMemberBinding="{Binding Path=GoldRetestUpper, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" ColumnGroupName="Gold"/>
<t:GridViewDataColumn Header="Test Lower (Min)"  HeaderTextAlignment="Center"
TextAlignment="Center" Style="{StaticResource TestLimits_ColumnStyle}"
DataMemberBinding="{Binding Path=GoldTestLower, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" ColumnGroupName="Gold"/>
<t:GridViewDataColumn Header="Test Upper (Max)" HeaderTextAlignment="Center"
TextAlignment="Center" Style="{StaticResource TestLimits_ColumnStyle}"
DataMemberBinding="{Binding Path=GoldTestUpper, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" ColumnGroupName="Gold"/>

<t:GridViewDataColumn Header="Test Control" HeaderTextAlignment="Center"
TextAlignment="Center" Style="{StaticResource TestLimits_ColumnStyle}"
DataMemberBinding="{Binding Path=GoldTestControl, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" ColumnGroupName="Gold"/>
<t:GridViewDataColumn Header="Linear Lower" HeaderTextAlignment="Center"
TextAlignment="Center" Style="{StaticResource TestLimits_ColumnStyle}"
DataMemberBinding="{Binding Path=GoldLinearLower, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" ColumnGroupName="Gold"/>
<t:GridViewDataColumn Header="Linear Upper" HeaderTextAlignment="Center"
TextAlignment="Center" Style="{StaticResource TestLimits_ColumnStyle}"
DataMemberBinding="{Binding Path=GoldLinearUpper, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" ColumnGroupName="Gold"/>
<t:GridViewDataColumn Header="Linear Control" HeaderTextAlignment="Center"
```

```
TextAlignment="Center" Style="{StaticResource TestLimits_ColumnStyle}"
DataMemberBinding="{Binding Path=GoldLinearControl, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" ColumnGroupName="Gold"/>
</t:RadGridView.Columns>

</t:RadGridView>

</Grid>
</UserControl>
```

OffsetLimits.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestLimits.OffsetLimits"
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:eSourceExt="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
xmlns:e="clr-namespace:MerlinTestStudio.DataModels;assembly=MerlinTestStudio.DataModels"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.TestLimits"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu>
<t:RadMenuItem Header="Add Test" IsEnabled="False"/>
<t:RadMenuItem Header="Remove Test"  IsEnabled="False"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Add Site" Command="{Binding Path=AddOffsetSiteCommand}" />
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=OffsetLimitsObj.Data.Count, StringFormat={}{0} Entries,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of tests in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="OffsetLimitsGridView" Grid.Row="1"
AutoGenerateColumns="False"
```

```
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Visible"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

Loaded="OffsetLimitsGridView_Loaded"
CellEditEnded="OffsetLimitsGridView_CellEditEnded"

RowHeight="30"
AllowDrop="False"
CanUserDeleteRows="True"
SelectedItem="{Binding Path=SelectedLimit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=OffsetLimitsObj.Data, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>

</t:RadGridView>

</Grid>
</UserControl>
```

TestFixtures.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestLimits.TestFixtures"
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:eSourceExt="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
xmlns:e="clr-namespace:MerlinTestStudio.DataModels;assembly=MerlinTestStudio.DataModels"
```

```
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.TestLimits"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu>
<t:RadMenuItem Header="Add Test" IsEnabled="False"/>
<t:RadMenuItem Header="Remove Test"  IsEnabled="False"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Add Site" Command="{Binding Path=AddTestFixtureSiteCommand}" />
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=TestFixturesObj.Data.Count, StringFormat={}{0} Entries,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of tests in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="TestFixturesGridView" Grid.Row="1"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Visible"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

Loaded="TestFixturesGridView_Loaded"
CellEditEnded="TestFixturesGridView_CellEditEnded"

RowHeight="30"
AllowDrop="False"
CanUserDeleteRows="True"
```

```
SelectedItem="{Binding Path=SelectedLimit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=TestFixturesObj.Data, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>

</t:RadGridView>

</Grid>
</UserControl>
```

TestLimits.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestLimits.TestLimits"
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:MerlinTestStudio_Demo_Telerik.UserControls.TestLimits"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800"
Loaded="UserControl_Loaded"
Unloaded="UserControl_Unloaded">

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu>
<t:RadMenuItem Header="Add Test" IsEnabled="False"/>
<t:RadMenuItem Header="Remove Test"  IsEnabled="False"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Add Multi-Pass Bin" Command="{Binding
Path=AddMultiPassBinCommand}" />
```

```
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Add Multi-Fail Bin" Command="{Binding
Path=AddMultiFailBinCommand}" />
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=TestLimitsObj.Data.Count, StringFormat={}{0} Entries,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of tests in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="TestLimitsGridView" Grid.Row="1"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Visible"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

Loaded="TestLimitsGridView_Loaded"
CellEditEnded="TestLimitsGridView_CellEditEnded"

RowHeight="30"
AllowDrop="False"
CanUserDeleteRows="True"
SelectedItem="{Binding Path=SelectedLimit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=TestLimitsObj.Data, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadGridView.Resources>
<!-- If you use NoXaml dlls and the implicit styles theming you will need to set also the BasedOn
property to the Style object-->

<!-- BasedOn="{StaticResource CommonHeaderPresenterStyle}" -->
<Style TargetType="t:CommonHeaderPresenter" >
<Setter Property="HorizontalContentAlignment" Value="Center"/>
<Setter Property="VerticalContentAlignment" Value="Center"/>
```

```
<Setter Property="IsHitTestVisible" Value="True"/>
</Style>

<!-- BasedOn="{StaticResource CommonColumnHeaderStyle}" -->
<Style TargetType="t:CommonColumnHeader">
<Setter Property="HorizontalContentAlignment" Value="Center"/>
<Setter Property="VerticalContentAlignment" Value="Center"/>
<Setter Property="IsHitTestVisible" Value="True"/>
</Style>
</t:RadGridView.Resources>

<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>

</t:RadGridView>

</Grid>
</UserControl>
```

TracelossLimits.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestLimits.TracelossLimits"
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:eSourceExt="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
xmlns:e="clr-namespace:MerlinTestStudio.DataModels;assembly=MerlinTestStudio.DataModels"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.UserControls.TestLimits"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadMenu>
```

```
<t:RadMenuItem Header="Add Test" IsEnabled="False"/>
<t:RadMenuItem Header="Remove Test"  IsEnabled="False"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Add Site" Command="{Binding Path=AddTracelossSiteCommand}" />
<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=TracelossLimitsObj.Data.Count, StringFormat={}{0} Entries,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of tests in this file / in the grid below."/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<t:RadGridView x:Name="TracelossLimitsGridView" Grid.Row="1"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="True"
CanUserSortColumns="True"
IsFilteringAllowed="False"
GroupRenderMode="Flat"

RowIndicatorVisibility="Visible"
IsSynchronizedWithCurrentItem ="True"
EnableColumnVirtualization="False"
EnableRowVirtualization="True"

Loaded="TracelossLimitsGridView_Loaded"
CellEditEnded="TracelossLimitsGridView_CellEditEnded"

RowHeight="30"
AllowDrop="False"
CanUserDeleteRows="True"
SelectedItem="{Binding Path=SelectedLimit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=TracelossLimitsObj.Data, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>
```

```
</t:RadGridView>

</Grid>
</UserControl>
```

DUT_End_Test.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestSequences.DUT_End_Test"
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:dd="clr-namespace:MerlinTestStudio_Demo_Telerik.Data"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.ViewModels"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:TemplateSelector="clr-
namespace:MerlinTestStudio_Demo_Telerik.Data.DataTemplateSelectors"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<TemplateSelector:DUTSequenceItemSelector x:Key="ActionSequenceItemSelecter" />
</UserControl.Resources>

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="4"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<!--Function Sequence And Function Library Grid-->
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="2"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Border Grid.Column="1" Grid.RowSpan="2" Background="DodgerBlue" Width="Auto"/>

<!--Function Sequence-->
<Grid Grid.Column="0" Margin="5">
```

```
<Grid.RowDefinitions>
<RowDefinition Height="30"  />
<RowDefinition Height="120" />
<RowDefinition Height="40"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Header-->
<Grid Grid.Row="0">
<TextBlock Text="Function Sequence" Style="{StaticResource FuctionSelectionListBoxHeader}"
/>
</Grid>

<!--Add Control Bar-->
<t:RadRibbonView Grid.Row="1" Width="Auto" Height="Auto"
TitleBarVisibility="Collapsed"
>
<t:RadRibbonView.Items>
<t:RadRibbonTab Header="Instruments">
<t:RadRibbonGroup Visibility="{Binding Path=DPS100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="DPS100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF100_Enabled}">
<t:RadButton  >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF100"                                      />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF110_Enabled}">
<t:RadButton >
<t:RadButton.Content>
```

```
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF110"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF200_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF200"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF300_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF300"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=HPA100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="HPA100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=SCAL100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
```

```
<TextBlock Text="SCAL100"                                />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=Keysight_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="Keysight"                               />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXI3025C_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="3025C"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXI3035C_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="3035C"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXIOpenATE_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="OpenATE"                                />
</StackPanel>
```

```
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
</t:RadRibbonTab>
<t:RadRibbonTab Header="Utilities">
<t:RadRibbonGroup>
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="Math"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
</t:RadRibbonTab>
</t:RadRibbonView.Items>
</t:RadRibbonView>

<!--Sequence Menu-->
<Grid Grid.Row="2">
<t:RadMenu>
<t:RadMenuItem Header="Duplicate" Command="{Binding
Path=MenuFunctionSequenceCommand}" CommandParameter="Duplicate"/>
<t:RadMenuSeparatorItem />
<t:RadMenuItem Header="Delete" Command="{Binding
Path=MenuFunctionSequenceCommand}" CommandParameter="Delete"/>
</t:RadMenu>
</Grid>

<!--Sequence List Box-->
<t:RadListBox x:Name="FunctionSequenceList" Grid.Row="3"
AllowDrop="True" EnableSelectionCaching="False"
ItemContainerStyle="{StaticResource NormListBoxItem}"
ItemsSource="{Binding Path=FunctionSequenceList, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=FSLFunctionSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Visibility="{Binding Path=ActionSelectedVisibility, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ItemTemplateSelector="{StaticResource ActionSequenceItemSelecter}">
<t:RadListBox.DragDropBehavior >
```

```
<dd:CustomDragDropBehaviorForFSL />
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
Path=FunctionSelectedCommand}"
RaiseOnHandledEvents="True" CommandParameter="FSL"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadListBox>
</Grid>

<!--Function Library-->
<Grid Grid.Column="3" Margin="5">
<Grid.RowDefinitions>
<RowDefinition Height="30"/>
<RowDefinition Height="120"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Header-->
<Grid Grid.Row="0">
<TextBlock Text="Function Library" Style="{StaticResource FuctionSelectionListBoxHeader}"/>
</Grid>

<!--Library Menu w/ function importer-->
<Grid Grid.Row="1">
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="2*"/>
</Grid.RowDefinitions>

<StackPanel Grid.Row="0">
<TextBlock Text="Source"/>
<t:RadComboBox x:Name="SourceComboBox"
ItemsSource="{Binding Path=ComboBoxDLLCollection, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedValue="{Binding Path=ComboBoxDLLSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Command="{Binding Path=DLLSelectedCommand}">
<t:RadComboBox.ItemTemplate>
```

```
<DataTemplate x:Name="cmbDTem">
<StackPanel Orientation="Horizontal">
<TextBlock Name="Item" Text="{Binding Path=NameOfDLL}" Width="100"/>
</StackPanel>
</DataTemplate>
</t:RadComboBox.ItemTemplate>
</t:RadComboBox>
</StackPanel>
<t:RadMenu Grid.Row="1">
<t:RadMenuItem Header="Import DLL" Command="{Binding Path=ImportDLLCommand}"/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>
</Grid>
<t:RadListBox x:Name="FunctionsLibraryList" Grid.Row="2"
ItemsSource="{Binding Path=FunctionLibrarySource, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemContainerStyle="{StaticResource DraggableListBoxItem}"
SelectedItem="{Binding Path=FLLFunctionSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
AllowDrop="False" EnableSelectionCaching="False"
>
<t:RadListBox.DragDropBehavior >
<dd:CustomDragDropBehaviorForFFL />
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
Path=FunctionSelectedCommand}"
RaiseOnHandledEvents="True" CommandParameter="FLL"/>
</t:EventToCommandBehavior.EventBindings>

</t:RadListBox>
</Grid>
</Grid>

<!--Vertical Grid Splitter-->
<GridSplitter Grid.Column="1" Width="4" HorizontalAlignment="Stretch" />

<!--Content Control Panel-->
<Grid Grid.Column="2">
```

```
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Switchable/Collapseable Headers-->
<Grid Grid.Row="0" Height="40">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<TextBlock Grid.Column="0" Text="Parameters/Controls" Style="{StaticResource
FuctionSelectionListBoxHeader}"/>

</Grid>

<!--Control/Parameters Viewer-->
<ContentPresenter Grid.Row="1" Content="{Binding Path=ControlPanelContent, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Height="Auto" Width="Auto">

</ContentPresenter>
</Grid>

</Grid>
</UserControl>
```

DUT_Start_Test.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestSequences.DUT_Start_Test"
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:dd="clr-namespace:MerlinTestStudio_Demo_Telerik.Data"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.ViewModels"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:TemplateSelector="clr-
namespace:MerlinTestStudio_Demo_Telerik.Data.DataTemplateSelectors"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
```

```
<TemplateSelector:DUTSequenceItemSelector x:Key="ActionSequenceItemSelecter" />
</UserControl.Resources>

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="4"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<!--Function Sequence And Function Library Grid-->
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="2"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Border Grid.Column="1" Grid.RowSpan="2" Background="DodgerBlue" Width="Auto"/>

<!--Function Sequence-->
<Grid Grid.Column="0" Margin="5">
<Grid.RowDefinitions>
<RowDefinition Height="30"  />
<RowDefinition Height="120" />
<RowDefinition Height="40"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Header-->
<Grid Grid.Row="0">
<TextBlock Text="Function Sequence" Style="{StaticResource FuctionSelectionListBoxHeader}"
/>
</Grid>

<!--Add Control Bar-->
<t:RadRibbonView Grid.Row="1" Width="Auto" Height="Auto"
TitleBarVisibility="Collapsed"
>
<t:RadRibbonView.Items>
<t:RadRibbonTab Header="Instruments">
<t:RadRibbonGroup Visibility="{Binding Path=DPS100_Enabled}">
<t:RadButton >
```

```
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="DPS100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF100_Enabled}">
<t:RadButton  >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF100"                                     />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding  Path=RF110_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF110"                                      />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF200_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF200"                                     />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF300_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
```

```
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF300"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=HPA100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="HPA100"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=SCAL100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="SCAL100"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=Keysight_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="Keysight"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXI3025C_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="3025C"                                  />
```

```
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXI3035C_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="3035C"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXIOpenATE_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="OpenATE"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
</t:RadRibbonTab>
<t:RadRibbonTab Header="Utilities">
<t:RadRibbonGroup>
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="Math"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
</t:RadRibbonTab>
</t:RadRibbonView.Items>
</t:RadRibbonView>

<!--Sequence Menu-->
<Grid Grid.Row="2">
```

```
<t:RadMenu>
<t:RadMenuItem Header="Duplicate" Command="{Binding
Path=MenuFunctionSequenceCommand}" CommandParameter="Duplicate"/>
<t:RadMenuSeparatorItem />
<t:RadMenuItem Header="Delete" Command="{Binding
Path=MenuFunctionSequenceCommand}" CommandParameter="Delete"/>
</t:RadMenu>
</Grid>

<!--Sequence List Box-->
<t:RadListBox x:Name="FunctionSequenceList" Grid.Row="3"
AllowDrop="True" EnableSelectionCaching="False"
ItemContainerStyle="{StaticResource NormListBoxItem}"
ItemsSource="{Binding Path=FunctionSequenceList, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=FSLFunctionSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Visibility="{Binding Path=ActionSelectedVisibility, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ItemTemplateSelector="{StaticResource ActionSequenceItemSelecter}">
<t:RadListBox.DragDropBehavior >
<dd:CustomDragDropBehaviorForFSL />
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
Path=FunctionSelectedCommand}"
RaiseOnHandledEvents="True" CommandParameter="FSL"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadListBox>
</Grid>

<!--Function Library-->
<Grid Grid.Column="3" Margin="5">
<Grid.RowDefinitions>
<RowDefinition Height="30"/>
<RowDefinition Height="120"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

```
<!--Header-->
<Grid Grid.Row="0">
<TextBlock Text="Function Library" Style="{StaticResource FuctionSelectionListBoxHeader}"/>
</Grid>

<!--Library Menu w/ function importer-->
<Grid Grid.Row="1">
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="2*"/>
</Grid.RowDefinitions>

<StackPanel Grid.Row="0">
<TextBlock Text="Source"/>
<t:RadComboBox x:Name="SourceComboBox"
ItemsSource="{Binding Path=ComboBoxDLLCollection, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedValue="{Binding Path=ComboBoxDLLSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Command="{Binding Path=DLLSelectedCommand}">
<t:RadComboBox.ItemTemplate>
<DataTemplate x:Name="cmbDTem">
<StackPanel Orientation="Horizontal">
<TextBlock Name="Item" Text="{Binding Path=NameOfDLL}" Width="100"/>
</StackPanel>
</DataTemplate>
</t:RadComboBox.ItemTemplate>
</t:RadComboBox>
</StackPanel>
<t:RadMenu Grid.Row="1">
<t:RadMenuItem Header="Import DLL" Command="{Binding Path=ImportDLLCommand}"/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>
</Grid>
<t:RadListBox x:Name="FunctionsLibraryList" Grid.Row="2"
ItemsSource="{Binding Path=FunctionLibrarySource, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemContainerStyle="{StaticResource DraggableListBoxItem}"
SelectedItem="{Binding Path=FLLFunctionSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
AllowDrop="False" EnableSelectionCaching="False"
>
```

```
<t:RadListBox.DragDropBehavior >
<dd:CustomDragDropBehaviorForFFL />
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
Path=FunctionSelectedCommand}"
RaiseOnHandledEvents="True" CommandParameter="FLL"/>
</t:EventToCommandBehavior.EventBindings>

</t:RadListBox>
</Grid>
</Grid>

<!--Vertical Grid Splitter-->
<GridSplitter Grid.Column="1" Width="4" HorizontalAlignment="Stretch" />

<!--Content Control Panel-->
<Grid Grid.Column="2">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Switchable/Collapseable Headers-->
<Grid Grid.Row="0" Height="40">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<TextBlock Grid.Column="0" Text="Parameters/Controls" Style="{StaticResource
FuctionSelectionListBoxHeader}"/>

</Grid>

<!--Control/Parameters Viewer-->
<ContentPresenter Grid.Row="1" Content="{Binding Path=ControlPanelContent, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Height="Auto" Width="Auto">

</ContentPresenter>
```

```
</Grid>

</Grid>
</UserControl>
```

DUT_Test.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestSequences.DUT_Test"
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:MerlinTestStudio_Demo_Telerik.ViewModels"
xmlns:SS="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.StyleSelectors"
xmlns:dd="clr-namespace:MerlinTestStudio_Demo_Telerik.Data"
xmlns:TemplateSelector="clr-
namespace:MerlinTestStudio_Demo_Telerik.Data.DataTemplateSelectors"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:TemplateSelector1="clr-
namespace:MerlinTestStudio_Demo_Telerik.Data.Selectors.DataTemplateSelectors"
mc:Ignorable="d"
Height="Auto"  Width="Auto" Loaded="UserControl_Loaded">

<UserControl.Resources>
<t:BooleanToVisibilityConverter x:Key="BooltoVisConverter"/>
<TemplateSelector:DUTSequenceItemSelector x:Key="ActionSequenceItemSelecter" />
<TemplateSelector1:ParameterTemplateSelector x:Key="ParameterValueEditorTemplateSelector"
/>

<!--ListBox Headers-->
<Style x:Key="FuctionSelectionListBoxHeader" TargetType="TextBlock">
<Setter Property="HorizontalAlignment" Value="Center"/>
<Setter Property="VerticalAlignment" Value="Center"/>
<Setter Property="FontSize" Value="20"/>
<Setter Property="Width" Value="Auto"/>
<Setter Property="Height" Value="Auto"/>
<Setter Property="Foreground" Value="LightGray"/>
</Style>
<Style x:Key="ParameterGridHeaders" TargetType="TextBlock">
<Setter Property="HorizontalAlignment" Value="Left"/>
<Setter Property="VerticalAlignment" Value="Center"/>
<Setter Property="FontSize" Value="16"/>
<Setter Property="Width" Value="Auto"/>
```

```
<Setter Property="Height" Value="Auto"/>
<Setter Property="Foreground" Value="LightGray"/>
</Style>

<!--Flipped Image Style-->
<Style TargetType="Image" x:Key="FlippedImage">
<Setter Property="RenderTransformOrigin" Value="0.5,0.5"/>
<Setter Property="RenderTransform">
<Setter.Value>
<TransformGroup>
<RotateTransform Angle="180"/>
</TransformGroup>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="MenuItemContainerStyle" TargetType="t:RadMenuItem">
<Setter Property="Header" Value="{Binding Path=Text}"          />
<Setter Property="ItemsSource" Value="{Binding Path=SubItems}"/>
<Setter Property="IsEnabled" Value="{Binding Path=IsEnabled}"/>
</Style>

<Style x:Key="BetterStyle" TargetType="t:RadListBoxItem">
<Setter Property="t:DragDropManager.AllowCapturedDrag" Value="True"/>
<Setter Property="t:DragDropManager.TouchDragTrigger" Value="TapAndHold"/>
</Style>

</UserControl.Resources>

<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"  />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"     />
<ColumnDefinition Width="4"    />
<ColumnDefinition Width="1.4*"/>
</Grid.ColumnDefinitions>

<!--DUT_Test Toolbar-->
<t:RadMenu Grid.Row="0" Grid.ColumnSpan="2">
```

```
<t:RadMenuItem  Padding="2" ToolTip="Moves the selected row up by 1" Command="{Binding Path=UpOrDownCommand}" CommandParameter="Up">
<t:RadMenuItem.Header>
<StackPanel>
<Image Source="/Resources/Images/MenuItems/SingleArrow.png" Width="30" Height="30"/>
</StackPanel>
</t:RadMenuItem.Header>
</t:RadMenuItem>
<t:RadMenuItem Padding="2" ToolTip="Moves the selected row down by 1" Command="{Binding Path=UpOrDownCommand}" CommandParameter="Down">
<t:RadMenuItem.Header>
<StackPanel>
<Image Source="/Resources/Images/MenuItems/SingleArrow.png" Width="30" Height="30" Style="{StaticResource FlippedImage}"/>
</StackPanel>
</t:RadMenuItem.Header>
</t:RadMenuItem>

<t:RadMenuSeparatorItem/>
<t:RadMenuItem x:Name="AddBlockBtn" Command="{Binding Path=AddBlockCommand}" ToolTip="Add a sequence object to the Test Sequence.">
<t:RadMenuItem.Header>
<t:RadGlyph Glyph="&#xe11e;" />
</t:RadMenuItem.Header>
</t:RadMenuItem>



<t:RadMenuSeparatorItem/>
<TextBlock Text="{Binding Path=ParentProject.SequenceItems.Count, StringFormat={}{0} Entries, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ToolTip="The number of sequence items in this file / in the grid below."/>

<t:RadMenuSeparatorItem/>

<t:RadMenuItem x:Name="ResyncBtn" Command="{Binding Path=CallRebindCommand}" ToolTip="Resync" >
<t:RadMenuItem.Header>
<t:RadGlyph Glyph="&#xe103;" />
</t:RadMenuItem.Header>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Compile" Padding="2" Command="{Binding
```

```
Path=CompileSequenceCommand}">
<t:RadMenuItem.Icon>
<t:RadGlyph Glyph="&#xe132;" Foreground="White"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadMenu>

<!-- MoveToTool MenuItem-->
<StackPanel Orientation="Horizontal" Grid.Row="0" Grid.ColumnSpan="2"
HorizontalAlignment="Right"  VerticalAlignment="Center" Margin="0 0 20 0">
<TextBlock Text="Move Selection at Sequence # :" VerticalAlignment="Center" Margin="0 0 8 0"/>
<t:RadNumericUpDown x:Name="LocationInput" Width="40" Padding="4"
KeyUp="LocationInput_KeyUp"
NumberDecimalDigits="0"
Minimum="1"
Maximum="{Binding ElementName=HierarchicalGridView, Path=ItemsSource.Count}">
<t:RadNumericUpDown.InputBindings>
<KeyBinding Command="{Binding Path=MoveSelectedToCommand}"
CommandParameter="{Binding ElementName=LocationInput, Path=Value}" Key="Enter" />
</t:RadNumericUpDown.InputBindings>
</t:RadNumericUpDown>
</StackPanel>

<!--DUT_Test GridView-->
<t:RadGridView x:Name="HierarchicalGridView"
Grid.Row="1" Grid.Column="0"
AutoGenerateColumns="False"
CanUserFreezeColumns="False"
ShowGroupPanel="False"
CanUserSortColumns="False"
CanUserDeleteRows="False"
IsFilteringAllowed="False"
RowIndicatorVisibility="Collapsed"
GroupRenderMode="Flat"
AllowDrop="True"

SelectionMode="Single"
SelectionUnit="FullRow"

DataLoadMode="Asynchronous"
EnableColumnVirtualization="True"
EnableRowVirtualization="True"
```

```
IsExpandableBinding="{Binding Path=IsExpandable, Mode=OneWay}"
IsExpandedBinding="{Binding Path=IsExpanded, Mode=TwoWay}"
SelectedItem="{Binding Path=Selected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=ParentProject.SequenceItems, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
Loaded="HierarchicalGridView_Loaded"
CellEditEnded="HierarchicalGridView_CellEditEnded">

<t:RadGridView.RowStyleSelector>
<SS:DUT_TestRowStyleSelector />
</t:RadGridView.RowStyleSelector>

<t:RadContextMenu.ContextMenu>
<t:RadContextMenu x:Name="GridContextMenu"
ItemContainerStyle="{StaticResource MenuItemContainerStyle}"
Opened="GridContextMenu_Opened"
ItemClick="GridContextMenu_ItemClick"/>
</t:RadContextMenu.ContextMenu>

<t:RadGridView.Resources>
<DataTemplate x:Key="DraggedItemTemplate">
<StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="Dragging:" />
<TextBlock Text="{Binding CurrentDraggedItem}"
FontWeight="Bold" />
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock Text="{Binding CurrentDropPosition}"
FontWeight="Bold"
MinWidth="45" />
<TextBlock Text=", ("
Foreground="Gray" />
<TextBlock Text="{Binding CurrentDraggedOverItem}" />
<TextBlock Text=")"
Foreground="Gray" />
</StackPanel>
</StackPanel>
</DataTemplate>
</t:RadGridView.Resources>
```

```
<t:RadGridView.HierarchyChildTemplate>
<DataTemplate>
<t:RadGridView  x:Name="ChildGrid"
ItemsSource="{Binding Path=Tests, Mode=OneWay, UpdateSourceTrigger=PropertyChanged}"
ShowGroupPanel="False"
AutoGenerateColumns="False">
<t:RadGridView.Columns>
<t:GridViewDataColumn Header="Test Number"
IsReadOnly="True"
DataMemberBinding="{Binding Path=TestNumber, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
CellStyle="{StaticResource CenteredCellStyle}"
DisplayIndex="0">
</t:GridViewDataColumn>
<t:GridViewDataColumn Header="Test Name"
DataMemberBinding="{Binding Path=TestName, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayIndex="1">
</t:GridViewDataColumn>
<t:GridViewComboBoxColumn Header="Result"
DataMemberBinding="{Binding Path=TestResult, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=DataContext.TestResultOptions, RelativeSource={RelativeSource
Mode=FindAncestor, AncestorType={x:Type t:RadGridView}}, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditorStyle="{StaticResource DropDownOnCellEdit}"
DisplayIndex="2">
</t:GridViewComboBoxColumn>
</t:RadGridView.Columns>
</t:RadGridView>
</DataTemplate>
</t:RadGridView.HierarchyChildTemplate>

<t:RadGridView.Columns>
<t:GridViewDataColumn Header="Sequence"
IsReadOnly="True"
DataMemberBinding="{Binding Path=Sequence, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
CellStyle="{StaticResource CenteredCellStyle}"
DisplayIndex="0">
</t:GridViewDataColumn>
```

```
<t:GridViewDataColumn Header="Test #"
DataMemberBinding="{Binding Path=TestNumber, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayIndex="1">
<t:GridViewDataColumn.Style>
<Style TargetType="t:GridViewDataColumn">

</Style>
</t:GridViewDataColumn.Style>
</t:GridViewDataColumn>
<t:GridViewDataColumn Header="Name"
DataMemberBinding="{Binding Path=ItemName, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayIndex="2">
</t:GridViewDataColumn>

<t:GridViewComboBoxColumn Header="Command"
DataMemberBinding="{Binding Path=Command, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=CommandStrings, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditTriggers="CellClick"
EditorStyle="{StaticResource DropDownOnCellEdit}"
DisplayIndex="3">
</t:GridViewComboBoxColumn>

<t:GridViewDataColumn Header="Function"
IsReadOnly="True"
DataMemberBinding="{Binding Path=Function, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
DisplayIndex="4">
</t:GridViewDataColumn>
<t:GridViewComboBoxColumn Header="Result"
DataMemberBinding="{Binding Path=Test.TestResult, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSourceBinding="{Binding Path=TestResultOptions, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EditorStyle="{StaticResource DropDownOnCellEdit}"
DisplayIndex="5">
</t:GridViewComboBoxColumn>

</t:RadGridView.Columns>
```

```
</t:RadGridView>

<!--Vertical Grid Splitter-->
<GridSplitter Width="4" Grid.Column="1" Grid.RowSpan="3" HorizontalAlignment="Stretch"/>

<!-- Function/Parameters Display-->
<Grid Grid.Column="2" Grid.RowSpan="2">
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*"  />
<RowDefinition Height="4" />
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Add Control Bar-->
<t:RadRibbonView Grid.Row="0" Width="Auto" Height="Auto" TitleBarVisibility="Collapsed"
Visibility="Collapsed">
<t:RadRibbonView.Items>
<t:RadRibbonTab Header="Instruments" >
<t:RadRibbonGroup Visibility="{Binding Path=DPS100_Enabled }" >
<t:RadButton Command="{Binding Path=AddControlCommand}" CommandParameter="DPS" >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="DPS"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF100_Enabled}">
<t:RadButton  >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF110_Enabled}">
<t:RadButton >
```

```
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF110"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF200_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF200"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF300_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF300"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=HPA100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="HPA100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=SCAL100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
```

```
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="SCAL100"                                   />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXIOpenATE_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="OpenATE"                                   />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
</t:RadRibbonTab>
<t:RadRibbonTab Header="Utilities">
<t:RadRibbonGroup>
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="Math"                                      />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
</t:RadRibbonTab>
</t:RadRibbonView.Items>
</t:RadRibbonView>
<Grid Grid.Row="0" Width="Auto" Height="Auto">
<t:RadMenu >
<t:RadMenuItem Header="Instruments">
<t:RadMenuItem/>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Utilities" >
<t:RadMenuItem Header="Math"/>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Header="Remove" Command="{Binding Path=RemoveFunctionCommand}"/>
```

```
</t:RadMenu>
</Grid>

<!--Sequence Item Display-->
<Grid Grid.Row="1">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="30"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<Grid Grid.Row="0" Grid.Column="0">
<TextBlock Text="Function Sequence" Style="{StaticResource FuctionSelectionListBoxHeader}"/>
</Grid>
<t:RadListBox x:Name="FunctionSequenceListDUT" Grid.Row="1" Grid.Column="0"
SelectedItem="{Binding Path=FunctionSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemContainerStyle="{StaticResource BetterStyle}"
ItemTemplateSelector="{StaticResource ActionSequenceItemSelecter}"
EnableSelectionCaching="False"
AllowDrop="True"
SelectionChanged="FunctionSequenceListDUT_SelectionChanged"
>

<t:RadListBox.Style>
<Style TargetType="t:RadListBox">
<Setter Property="ItemsSource" Value="{Binding ElementName=HierarchicalGridView,
Path=SelectedItem.SequenceComponents}"/>
<Setter Property="IsEnabled" Value="True"/>
<Style.Triggers>
<DataTrigger Binding="{Binding ElementName=HierarchicalGridView, Path=SelectedItem}"
Value="">
<Setter Property="IsEnabled" Value="False"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadListBox.Style>
```

```
<t:RadListBox.DragDropBehavior>
<dd:CustomDragDropBehaviorForFSL AllowReorder="True"/>
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>

</t:RadListBox>

<Grid Grid.Row="0" Grid.Column="1">
<TextBlock Text="Function Library" Style="{StaticResource FuctionSelectionListBoxHeader}"/>
</Grid>
<Grid Grid.Row="1" Grid.Column="1">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="3*"/>
</Grid.RowDefinitions>

<t:RadMenu Grid.Row="0">
<t:RadMenuItem Header="Import DLL" Command="{Binding Path=ImportDLLCommand}"/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<StackPanel Grid.Row="1" Margin="5">
<TextBlock Text="Source: "/>
<t:RadComboBox x:Name="SourceComboBox"
ItemsSource="{Binding Path=ParentProject.DLLs, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedValue="{Binding Path=ComboBoxDLLSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">
<t:RadComboBox.ItemTemplate>
<DataTemplate x:Name="cmbDTem">
<StackPanel Orientation="Horizontal">
<TextBlock Name="Item" Text="{Binding Path=NameOfDLL}" Width="Auto"/>
</StackPanel>
</DataTemplate>
</t:RadComboBox.ItemTemplate>
</t:RadComboBox>
</StackPanel>

<t:RadListBox x:Name="FunctionsLibraryList"
```

```
Grid.Row="2"
ItemContainerStyle="{StaticResource DraggableListBoxItem}"
AllowDrop="True"
EnableSelectionCaching="False">

<t:RadListBox.Style>
<Style TargetType="t:RadListBox">
<Setter Property="ItemsSource" Value="{Binding ElementName=SourceComboBox,
Path=SelectedValue.DLLFunctions}"/>
</Style>
</t:RadListBox.Style>

<t:RadListBox.DragDropBehavior >
<dd:CustomDragDropBehaviorForFFL AllowReorder="False" />
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
Path=FunctionSelectedCommand}"
RaiseOnHandledEvents="True" CommandParameter="FLL"/>
</t:EventToCommandBehavior.EventBindings>

</t:RadListBox>

</Grid>

</Grid>

<!-- Horizontal Splitter-->
<GridSplitter Height="4" Grid.Row="2" HorizontalAlignment="Stretch" ResizeDirection="Rows"/>

<!--Content Control Panel-->
<Grid Grid.Row="3">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<Grid Grid.Row="0">
<StackPanel Height="Auto" Width="Auto"
```

```
VerticalAlignment="Center"
HorizontalAlignment="Right"
Orientation="Horizontal"
>
<Border BorderBrush="White" BorderThickness="1" Margin="10">
<t:RadToggleSwitchButton x:Name="ValMapToggleButton"
Margin="1"
ContentPosition="Both"
CheckedContent="Value"
UncheckedContent="Map"
Checked="ValMapToggleButton_Checked">
<t:RadToggleSwitchButton.Style>
<Style TargetType="t:RadToggleSwitchButton">
<Setter Property="IsEnabled" Value="True"/>
<Style.Triggers>
<DataTrigger Binding="{Binding Path=Selected.ItemType, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Value="2">
<Setter Property="IsChecked" Value="True"/>
<Setter Property="IsEnabled" Value="False"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadToggleSwitchButton.Style>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="Checked" Command="{Binding
Path=RefreshParameterValuesCommand}" RaiseOnHandledEvents="True" />
</t:EventToCommandBehavior.EventBindings>
</t:RadToggleSwitchButton>
</Border>
</StackPanel>
<StackPanel Height="Auto" Width="Auto"
VerticalAlignment="Center"
HorizontalAlignment="Left"
Orientation="Horizontal"
>
<t:RadButton x:Name="AutoMapBtn" Content="AutoMap" Margin="5" Padding="5"
Command="{Binding Path=OpenAutoMapCommand}"/>
<t:RadButton x:Name="ParamValRefreshBtn" Content="Refresh" Margin="5" Padding="5"
Command="{Binding Path=RefreshParameterValuesCommand}" IsEnabled="{Binding
ElementName=ValMapToggleButton, Path=IsChecked, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"/>
</StackPanel>
```

```
</Grid>

<!-- Parameters Listbox -->
<t:RadListBox x:Name="ParametersValuesList" Grid.Row="1" Margin="0 8 0 0"
ItemsSource="{Binding ElementName=FunctionSequenceListDUT,
Path=SelectedItem.FunctionParameters, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
EnableSelectionCaching="False">
<t:RadListBox.Style>
<Style TargetType="t:RadListBox">
<Setter Property="Visibility" Value="Visible"/>
<Style.Triggers>
<DataTrigger Binding="{Binding ElementName=ValMapToggleButton, Path=IsChecked,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}" Value="True">
<Setter Property="ItemTemplateSelector" Value="{StaticResource
ParameterValueEditorTemplateSelector}"/>
</DataTrigger>
<DataTrigger Binding="{Binding ElementName=ValMapToggleButton, Path=IsChecked,
Mode=OneWay, UpdateSourceTrigger=PropertyChanged}" Value="False">
<Setter Property="ItemTemplate" Value="{StaticResource MappingPointerTemplate}"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadListBox.Style>
</t:RadListBox>

<!-- NOT CURRENTLY IN USE  -->
<ContentPresenter Grid.Row="1" Visibility="Collapsed"
Content="{Binding Path=ControlPanelContent, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
Height="Auto"
Width="Auto">
</ContentPresenter>
</Grid>

</Grid>
</Grid>
</UserControl>
```

SessionLoad.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestSequences.Session_Load"
```

```
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:dd="clr-namespace:MerlinTestStudio_Demo_Telerik.Data"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.ViewModels"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:TemplateSelector="clr-
namespace:MerlinTestStudio_Demo_Telerik.Data.DataTemplateSelectors"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<TemplateSelector:DUTSequenceItemSelector x:Key="ActionSequenceItemSelecter" />
</UserControl.Resources>

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="4"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<!--Function Sequence And Function Library Grid-->
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="2"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Border Grid.Column="1" Grid.RowSpan="2" Background="DodgerBlue" Width="Auto"/>

<!--Function Sequence-->
<Grid Grid.Column="0" Margin="5">
<Grid.RowDefinitions>
<RowDefinition Height="30"  />
<RowDefinition Height="120" />
<RowDefinition Height="40"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Header-->
```

```
<Grid Grid.Row="0">
<TextBlock Text="Function Sequence" Style="{StaticResource FuctionSelectionListBoxHeader}"
/>
</Grid>

<!--Add Control Bar-->
<t:RadRibbonView Grid.Row="1" Width="Auto" Height="Auto"
TitleBarVisibility="Collapsed"
>
<t:RadRibbonView.Items>
<t:RadRibbonTab Header="Instruments">
<t:RadRibbonGroup Visibility="{Binding Path=DPS100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="DPS100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF100_Enabled}">
<t:RadButton  >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF100"                                     />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF110_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF110"                                     />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF200_Enabled}">
```

```
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF200"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF300_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF300"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=HPA100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="HPA100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=SCAL100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="SCAL100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=Keysight_Enabled}">
<t:RadButton >
<t:RadButton.Content>
```

```
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="Keysight"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXI3025C_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="3025C"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXI3035C_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="3035C"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXIOpenATE_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="OpenATE"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
</t:RadRibbonTab>
<t:RadRibbonTab Header="Utilities">
<t:RadRibbonGroup>
<t:RadButton >
<t:RadButton.Content>
```

```
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="Math"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
</t:RadRibbonTab>
</t:RadRibbonView.Items>
</t:RadRibbonView>

<!--Sequence Menu-->
<Grid Grid.Row="2">
<t:RadMenu>
<t:RadMenuItem Header="Duplicate" Command="{Binding
Path=MenuFunctionSequenceCommand}" CommandParameter="Duplicate"/>
<t:RadMenuSeparatorItem />
<t:RadMenuItem Header="Delete" Command="{Binding
Path=MenuFunctionSequenceCommand}" CommandParameter="Delete"/>
</t:RadMenu>
</Grid>

<!--Sequence List Box-->
<t:RadListBox x:Name="FunctionSequenceList" Grid.Row="3"
AllowDrop="True" EnableSelectionCaching="False"
ItemContainerStyle="{StaticResource NormListBoxItem}"
ItemsSource="{Binding Path=FunctionSequenceList, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=FSLFunctionSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Visibility="{Binding Path=ActionSelectedVisibility, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ItemTemplateSelector="{StaticResource ActionSequenceItemSelecter}">
<t:RadListBox.DragDropBehavior >
<dd:CustomDragDropBehaviorForFSL />
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
Path=FunctionSelectedCommand}"
```

```
RaiseOnHandledEvents="True" CommandParameter="FSL"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadListBox>
</Grid>

<!--Function Library-->
<Grid Grid.Column="3" Margin="5">
<Grid.RowDefinitions>
<RowDefinition Height="30"/>
<RowDefinition Height="120"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Header-->
<Grid Grid.Row="0">
<TextBlock Text="Function Library" Style="{StaticResource FuctionSelectionListBoxHeader}"/>
</Grid>

<!--Library Menu w/ function importer-->
<Grid Grid.Row="1">
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="2*"/>
</Grid.RowDefinitions>

<StackPanel Grid.Row="0">
<TextBlock Text="Source"/>
<t:RadComboBox x:Name="SourceComboBox"
ItemsSource="{Binding Path=ComboBoxDLLCollection, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedValue="{Binding Path=ComboBoxDLLSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Command="{Binding Path=DLLSelectedCommand}">
<t:RadComboBox.ItemTemplate>
<DataTemplate x:Name="cmbDTem">
<StackPanel Orientation="Horizontal">
<TextBlock Name="Item" Text="{Binding Path=NameOfDLL}" Width="100"/>
</StackPanel>
</DataTemplate>
</t:RadComboBox.ItemTemplate>
</t:RadComboBox>
</StackPanel>
```

```
<t:RadMenu Grid.Row="1">
<t:RadMenuItem Header="Import DLL" Command="{Binding Path=ImportDLLCommand}"/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>
</Grid>
<t:RadListBox x:Name="FunctionsLibraryList" Grid.Row="2"
ItemsSource="{Binding Path=FunctionLibrarySource, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemContainerStyle="{StaticResource DraggableListBoxItem}"
SelectedItem="{Binding Path=FLLFunctionSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
AllowDrop="False" EnableSelectionCaching="False"
>
<t:RadListBox.DragDropBehavior >
<dd:CustomDragDropBehaviorForFFL />
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
Path=FunctionSelectedCommand}"
RaiseOnHandledEvents="True" CommandParameter="FLL"/>
</t:EventToCommandBehavior.EventBindings>

</t:RadListBox>
</Grid>
</Grid>

<!--Vertical Grid Splitter-->
<GridSplitter Grid.Column="1" Width="4" HorizontalAlignment="Stretch" />

<!--Content Control Panel-->
<Grid Grid.Column="2">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Switchable/Collapseable Headers-->
<Grid Grid.Row="0" Height="40">
<Grid.ColumnDefinitions>
```

```
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<TextBlock Grid.Column="0" Text="Parameters/Controls" Style="{StaticResource
FuctionSelectionListBoxHeader}"/>

</Grid>

<!--Control/Parameters Viewer-->
<ContentPresenter Grid.Row="1" Content="{Binding Path=ControlPanelContent, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Height="Auto" Width="Auto">

</ContentPresenter>
</Grid>

</Grid>
</UserControl>
```

SessionUnload.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.TestSequences.SessionUnload"
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:dd="clr-namespace:MerlinTestStudio_Demo_Telerik.Data"
xmlns:local="clr-namespace:MerlinTestStudio_Demo_Telerik.ViewModels"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:TemplateSelector="clr-
namespace:MerlinTestStudio_Demo_Telerik.Data.DataTemplateSelectors"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<TemplateSelector:DUTSequenceItemSelector x:Key="ActionSequenceItemSelecter" />
</UserControl.Resources>

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="4"/>
<ColumnDefinition Width="*"/>
```

```
</Grid.ColumnDefinitions>

<!--Function Sequence And Function Library Grid-->
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="2"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Border Grid.Column="1" Grid.RowSpan="2" Background="DodgerBlue" Width="Auto"/>

<!--Function Sequence-->
<Grid Grid.Column="0" Margin="5">
<Grid.RowDefinitions>
<RowDefinition Height="30"  />
<RowDefinition Height="120" />
<RowDefinition Height="40"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Header-->
<Grid Grid.Row="0">
<TextBlock Text="Function Sequence" Style="{StaticResource FuctionSelectionListBoxHeader}"
/>
</Grid>

<!--Add Control Bar-->
<t:RadRibbonView Grid.Row="1" Width="Auto" Height="Auto"
TitleBarVisibility="Collapsed"
>
<t:RadRibbonView.Items>
<t:RadRibbonTab Header="Instruments">
<t:RadRibbonGroup Visibility="{Binding Path=DPS100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="DPS100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
```

```
<t:RadRibbonGroup Visibility="{Binding Path=RF100_Enabled}">
<t:RadButton  >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF110_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF110"                                     />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF200_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF200"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=RF300_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="RF300"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=HPA100_Enabled}">
<t:RadButton >
```

```
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="HPA100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=SCAL100_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="SCAL100"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=Keysight_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="Keysight"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXI3025C_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="3025C"                                    />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXI3035C_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
```

```
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="3035C"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Visibility="{Binding Path=PXIOpenATE_Enabled}">
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="OpenATE"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
</t:RadRibbonTab>
<t:RadRibbonTab Header="Utilities">
<t:RadRibbonGroup>
<t:RadButton >
<t:RadButton.Content>
<StackPanel >
<Image Source="/Resources/Images/MenuItems/AddIcon.png" Width="25" Height="25"/>
<TextBlock Text="Math"                                  />
</StackPanel>
</t:RadButton.Content>
</t:RadButton>
</t:RadRibbonGroup>
</t:RadRibbonTab>
</t:RadRibbonView.Items>
</t:RadRibbonView>

<!--Sequence Menu-->
<Grid Grid.Row="2">
<t:RadMenu>
<t:RadMenuItem Header="Duplicate" Command="{Binding
Path=MenuFunctionSequenceCommand}" CommandParameter="Duplicate"/>
<t:RadMenuSeparatorItem />
<t:RadMenuItem Header="Delete" Command="{Binding
Path=MenuFunctionSequenceCommand}" CommandParameter="Delete"/>
</t:RadMenu>
</Grid>
```

```
<!--Sequence List Box-->
<t:RadListBox x:Name="FunctionSequenceList" Grid.Row="3"
AllowDrop="True" EnableSelectionCaching="False"
ItemContainerStyle="{StaticResource NormListBoxItem}"
ItemsSource="{Binding Path=FunctionSequenceList, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=FSLFunctionSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Visibility="{Binding Path=ActionSelectedVisibility, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}"
ItemTemplateSelector="{StaticResource ActionSequenceItemSelecter}">
<t:RadListBox.DragDropBehavior >
<dd:CustomDragDropBehaviorForFSL />
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
Path=FunctionSelectedCommand}"
RaiseOnHandledEvents="True" CommandParameter="FSL"/>
</t:EventToCommandBehavior.EventBindings>
</t:RadListBox>
</Grid>

<!--Function Library-->
<Grid Grid.Column="3" Margin="5">
<Grid.RowDefinitions>
<RowDefinition Height="30"/>
<RowDefinition Height="120"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Header-->
<Grid Grid.Row="0">
<TextBlock Text="Function Library" Style="{StaticResource FuctionSelectionListBoxHeader}"/>
</Grid>

<!--Library Menu w/ function importer-->
<Grid Grid.Row="1">
<Grid.RowDefinitions>
```

```
<RowDefinition Height="*"/>
<RowDefinition Height="2*"/>
</Grid.RowDefinitions>

<StackPanel Grid.Row="0">
<TextBlock Text="Source"/>
<t:RadComboBox x:Name="SourceComboBox"
ItemsSource="{Binding Path=ComboBoxDLLCollection, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
SelectedValue="{Binding Path=ComboBoxDLLSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
Command="{Binding Path=DLLSelectedCommand}">
<t:RadComboBox.ItemTemplate>
<DataTemplate x:Name="cmbDTem">
<StackPanel Orientation="Horizontal">
<TextBlock Name="Item" Text="{Binding Path=NameOfDLL}" Width="100"/>
</StackPanel>
</DataTemplate>
</t:RadComboBox.ItemTemplate>
</t:RadComboBox>
</StackPanel>
<t:RadMenu Grid.Row="1">
<t:RadMenuItem Header="Import DLL" Command="{Binding Path=ImportDLLCommand}"/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>
</Grid>
<t:RadListBox x:Name="FunctionsLibraryList" Grid.Row="2"
ItemsSource="{Binding Path=FunctionLibrarySource, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemContainerStyle="{StaticResource DraggableListBoxItem}"
SelectedItem="{Binding Path=FLLFunctionSelected, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
AllowDrop="False" EnableSelectionCaching="False"
>
<t:RadListBox.DragDropBehavior >
<dd:CustomDragDropBehaviorForFFL />
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="SelectionChanged" Command="{Binding
```

```
Path=FunctionSelectedCommand}"
RaiseOnHandledEvents="True" CommandParameter="FLL"/>
</t:EventToCommandBehavior.EventBindings>

</t:RadListBox>
</Grid>
</Grid>

<!--Vertical Grid Splitter-->
<GridSplitter Grid.Column="1" Width="4" HorizontalAlignment="Stretch" />

<!--Content Control Panel-->
<Grid Grid.Column="2">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!--Switchable/Collapseable Headers-->
<Grid Grid.Row="0" Height="40">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<TextBlock Grid.Column="0" Text="Parameters/Controls" Style="{StaticResource
FuctionSelectionListBoxHeader}"/>

</Grid>

<!--Control/Parameters Viewer-->
<ContentPresenter Grid.Row="1" Content="{Binding Path=ControlPanelContent, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}" Height="Auto" Width="Auto">

</ContentPresenter>
</Grid>

</Grid>
</UserControl>
```

AutoGeneratedChecker.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Tools.AutoGeneratedChecker"
```

```
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:MerlinTestStudio_Demo_Telerik.UserControls.Tools"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="400">

<UserControl.Resources>
<Style x:Key="SelectionListBoxHeader" TargetType="TextBlock">
<Setter Property="HorizontalAlignment" Value="Center"/>
<Setter Property="VerticalAlignment" Value="Center"/>
<Setter Property="FontSize" Value="20"/>
<Setter Property="Width" Value="Auto"/>
<Setter Property="Height" Value="Auto"/>
<Setter Property="Foreground" Value="LightGray"/>
</Style>
</UserControl.Resources>

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="40"/>
<RowDefinition Height="*"/>
<RowDefinition Height="40"/>
</Grid.RowDefinitions>

<Grid Grid.Row="0" Grid.ColumnSpan="2">
<TextBlock Text="Parameters" Style="{StaticResource SelectionListBoxHeader}"/>
</Grid>

<t:RadListBox x:Name="ParameterMapList" Grid.Row="1" Grid.ColumnSpan="2" Margin="10"
ItemsSource="{Binding Path=AutoGeneratedParameters, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadListBox.ItemTemplate>
<DataTemplate>
<Grid>
```

```
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="2*"/>
</Grid.ColumnDefinitions>

<CheckBox Grid.Column="0" Content="Import" Foreground="White" FontSize="14"
IsChecked="{Binding Path=IsAutoGen, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" />
<TextBlock Grid.Column="1" Text="{Binding Path=ColumnName}" FontSize="14"/>

</Grid>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>

<!--Buttons Area-->
<WrapPanel Grid.Row="2" Grid.ColumnSpan="2" HorizontalAlignment="Center">
<t:RadButton Content="Cancel" Margin="0 0 40 0" Width="140" Height="30"
Command="{Binding Path=CancelBtnCommand}"
CommandParameter="{Binding RelativeSource={RelativeSource
AncestorType=t:RadWindow,Mode=FindAncestor}}">
</t:RadButton>
<t:RadButton Content="Done" Width="140" Height="30"
Command="{Binding Path=DoneBtnCommand}"
CommandParameter="{Binding RelativeSource={RelativeSource
AncestorType=t:RadWindow,Mode=FindAncestor}}">
</t:RadButton>
</WrapPanel>
</Grid>
</UserControl>
```

DataFormattingTool.xaml

```
<UserControl x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Tools.DataFormattingTool"
x:Name="DataFormattingToolControl"
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:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:Controls="clr-
namespace:Telerik.Windows.Controls.Spreadsheet.Controls;assembly=Telerik.Windows.Controls.
Spreadsheet"
```

```
xmlns:spreadsheet="clr-
namespace:Telerik.Windows.Controls.Spreadsheet;assembly=Telerik.Windows.Controls.Spreads
heet"
xmlns:Txt="clr-
namespace:Telerik.Windows.Documents.Spreadsheet.FormatProviders.TextBased.Txt;assembly=
Telerik.Windows.Documents.Spreadsheet"
xmlns:Csv="clr-
namespace:Telerik.Windows.Documents.Spreadsheet.FormatProviders.TextBased.Csv;assembly
=Telerik.Windows.Documents.Spreadsheet"
xmlns:Pdf="clr-
namespace:Telerik.Windows.Documents.Spreadsheet.FormatProviders.Pdf;assembly=Telerik.Wi
ndows.Documents.Spreadsheet.FormatProviders.Pdf"
xmlns:Xlsx="clr-
namespace:Telerik.Windows.Documents.Spreadsheet.FormatProviders.OpenXml.Xlsx;assembly=
Telerik.Windows.Documents.Spreadsheet.FormatProviders.OpenXml"
xmlns:helpers="clr-namespace:MerlinTestStudio_Demo_Telerik.Data.Helpers"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>

</UserControl.Resources>

<Grid x:Name="spreadsheetLayoutRoot" >
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="3*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<!--Ribbon Menu-->
<t:RadRibbonView x:Name="ribbonView" BackstageClippingElement="{Binding
ElementName=spreadsheetLayoutRoot}" DataContext="{Binding CommandDescriptors,
ElementName=radSpreadsheet}" HeaderBackground="LightGray" Title="{Binding
Workbook.Name, ElementName=radSpreadsheet}" Grid.Row="0" Grid.ColumnSpan="2">
<t:RadRibbonView.Resources>
<spreadsheet:FunctionsProvider x:Key="FunctionsProvider"/>
```

```
<t:IconSources x:Key="IconPaths"
DarkBasePath="/Telerik.Windows.Controls.Spreadsheet;component/Images/Dark/"
LightBasePath="/Telerik.Windows.Controls.Spreadsheet;component/Images/Light/"/>
<t:BooleanToVisibilityConverter x:Key="BoolToVisibilityValueConverter"/>
</t:RadRibbonView.Resources>
<t:RadRibbonView.ApplicationButtonContent>
<TextBlock Foreground="White" Text="File"/>
</t:RadRibbonView.ApplicationButtonContent>
<t:RadRibbonView.Backstage>
<t:RadRibbonBackstage>
<t:RadRibbonBackstageItem CloseOnClick="True" Command="{Binding NewFile.Command}"
Header="New" Icon="{t:IconResource IconRelativePath=16/new.png,
IconSources={StaticResource IconPaths}}" IsEnabled="{Binding NewFile.IsEnabled}"
IsSelectable="False"/>
<t:RadRibbonBackstageItem CloseOnClick="True" Command="{Binding OpenFile.Command}"
Header="Open" Icon="{t:IconResource IconRelativePath=16/open.png,
IconSources={StaticResource IconPaths}}" IsEnabled="{Binding OpenFile.IsEnabled}"
IsSelectable="False"/>
<t:RadRibbonBackstageItem CloseOnClick="True" Command="{Binding SaveFile.Command}"
Header="Save" Icon="{t:IconResource IconRelativePath=16/save.png,
IconSources={StaticResource IconPaths}}" IsEnabled="{Binding SaveFile.IsEnabled}"
IsSelectable="False"/>
<t:RadRibbonBackstageItem IsGroupSeparator="True"/>
<t:RadRibbonBackstageItem Header="Print" IsDefault="false">
<Controls:PrintPreviewControl RadSpreadsheet="{Binding RadSpreadsheet, Mode=OneTime}"/>
</t:RadRibbonBackstageItem>
</t:RadRibbonBackstage>
</t:RadRibbonView.Backstage>
<t:RadRibbonView.ContextualGroups>
<t:RadRibbonContextualGroup x:Name="PictureTools" Header="Picture Tools" IsActive="{Binding
PictureToolsTab.IsEnabled, Mode=OneWay}"/>
</t:RadRibbonView.ContextualGroups>
<t:RadRibbonView.QuickAccessToolBar>
<t:QuickAccessToolBar>
<t:RadRibbonButton Command="{Binding NewFile.Command}" Size="Small"
SmallImage="{t:IconResource IconRelativePath=16/new.png, IconSources={StaticResource
IconPaths}}" IsEnabled="{Binding NewFile.IsEnabled}" Text="New"/>
<t:RadRibbonButton Command="{Binding OpenFile.Command}"  Size="Small"
SmallImage="{t:IconResource IconRelativePath=16/open.png, IconSources={StaticResource
IconPaths}}" IsEnabled="{Binding OpenFile.IsEnabled}" Text="Open"/>
<t:RadRibbonButton Command="{Binding SaveFile.Command}" IsEnabled="{Binding
SaveFile.IsEnabled}" Size="Small" SmallImage="{t:IconResource IconRelativePath=16/save.png,
```

```
IconSources={StaticResource IconPaths}}" Text="Save"/>
<t:RadRibbonButton Command="{Binding Undo.Command}" IsEnabled="{Binding
Undo.IsEnabled}" Size="Small" SmallImage="{t:IconResource IconRelativePath=16/undo.png,
IconSources={StaticResource IconPaths}}" Text="Undo"/>
<t:RadRibbonButton Command="{Binding Redo.Command}" IsEnabled="{Binding
Redo.IsEnabled}" Size="Small" SmallImage="{t:IconResource IconRelativePath=16/redo.png,
IconSources={StaticResource IconPaths}}" Text="Redo"/>
</t:QuickAccessToolBar>
</t:RadRibbonView.QuickAccessToolBar>
<t:RadRibbonTab Header="Home" >
<t:RadRibbonGroup Header="Clipboard" IsEnabled="{Binding ClipboardGroup.IsEnabled}"
t:ScreenTip.Title="Clipboard">
<t:RadRibbonGroup.Variants>
<t:GroupVariant Priority="0" Variant="Large"/>
</t:RadRibbonGroup.Variants>
<t:RadRibbonSplitButton x:Name="PasteButton" Command="{Binding Paste.Command}"
t:ScreenTip.Description="Paste the contents of the Clipboard." IsEnabled="{Binding
Paste.IsEnabled}" LargeImage="{t:IconResource IconRelativePath=32/pasteActive.png,
IconSources={StaticResource IconPaths}}" Size="Large" Text="Paste" t:ScreenTip.Title="Paste
(Ctrl+V)">
<t:RadRibbonSplitButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuGroupItem BorderThickness="0" Background="Transparent" Header="Paste">
<t:RadMenuItem Command="{Binding Paste.Command}" Header="Paste" IsEnabled="{Binding
Paste.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/pasteNormal.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem CommandParameter="Formulas" Command="{Binding
PasteFormulas.Command}" Header="Formulas" IsEnabled="{Binding PasteFormulas.IsEnabled,
Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/pasteFormulas.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem CommandParameter="FormulasAndNumberFormats" Command="{Binding
PasteFormulasAndNumberFormats.Command}" Header="Formulas &amp;amp; Number Formatting"
IsEnabled="{Binding PasteFormulasAndNumberFormats.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
```

```
<Image Source="{t:IconResource IconRelativePath=16/pasteFormulasNumberFormatting.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem CommandParameter="ColumnWidths" Command="{Binding
PasteColumnWidths.Command}" Header="Column Widths" IsEnabled="{Binding
PasteColumnWidths.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/pasteKeepSourceColumnWidths.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadMenuGroupItem>
<t:RadMenuGroupItem BorderThickness="0" Background="Transparent" Header="Paste Values"
Margin="-1">
<t:RadMenuItem CommandParameter="Values" Command="{Binding PasteValues.Command}"
Header="Values" IsEnabled="{Binding PasteValues.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/pasteValues.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem CommandParameter="ValuesAndNumberFormats" Command="{Binding
PasteValuesAndNumberFormats.Command}" Header="Values &amp;amp; Number Formatting"
IsEnabled="{Binding PasteValuesAndNumberFormats.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/pasteValuesNumberFormatting.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadMenuGroupItem>
<t:RadMenuGroupItem BorderThickness="0" Background="Transparent" Header="Other Paste
Options" Margin="-1">
<t:RadMenuItem CommandParameter="Formats" Command="{Binding
PasteFormatting.Command}" Header="Formatting" IsEnabled="{Binding
PasteFormatting.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/pasteFormatting.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadMenuGroupItem>
```

```
</t:RadContextMenu>
</t:RadRibbonSplitButton.DropDownContent>
</t:RadRibbonSplitButton>
<StackPanel>
<t:RadRibbonButton Command="{Binding Cut.Command}" t:ScreenTip.Description="Cut the selection and put it on the Clipboard." IsEnabled="{Binding Cut.IsEnabled}" Size="Medium" SmallImage="{t:IconResource IconRelativePath=16/cut.png, IconSources={StaticResource IconPaths}}" Text="Cut" t:ScreenTip.Title="Cut"/>
<t:RadRibbonButton Command="{Binding Copy.Command}" t:ScreenTip.Description="Copy the selection and put it on the Clipboard." IsEnabled="{Binding Copy.IsEnabled}" Size="Medium" SmallImage="{t:IconResource IconRelativePath=16/copy.png, IconSources={StaticResource IconPaths}}" Text="Copy" t:ScreenTip.Title="Copy"/>
</StackPanel>
</t:RadRibbonGroup>
<t:RadRibbonGroup DialogLauncherVisibility="{Binding ShowFormatCellsDialog.IsEnabled, Converter={StaticResource BoolToVisibilityValueConverter}}" DialogLauncherCommandParameter="Font" t:ScreenTip.Description="Show the Font tab of the Format Cells dialog box." DialogLauncherCommand="{Binding ShowFormatCellsDialog.Command}" Header="Font" IsEnabled="{Binding FontGroup.IsEnabled}" t:ScreenTip.Title="Format Cells: Font">
<t:RadRibbonGroup.Variants>
<t:GroupVariant Priority="0" Variant="Large"/>
</t:RadRibbonGroup.Variants>
<t:RadOrderedWrapPanel>
<StackPanel Orientation="Horizontal">
<StackPanel.Resources>
<Controls:UnitToDipConverter x:Key="unitToDipConverter"/>
<Controls:DipToUnitConverter x:Key="dipToUnitConverter"/>
<Controls:ThemeFontTypeToTextConverter x:Key="themeFontTypeToTextConverter"/>
</StackPanel.Resources>
<t:RadRibbonComboBox CanAutocompleteSelectItems="False" CanKeyboardNavigationSelectItems="False" t:ScreenTip.Description="Change the font family." ScrollViewer.HorizontalScrollBarVisibility="Disabled" IsEditable="False" ItemsSource="{Binding FontsProvider.RegisteredFonts, ElementName=radSpreadsheet}" IsEnabled="{Binding SetFontFamily.IsEnabled}" MaxDropDownHeight="400" SelectedValue="{Binding SetFontFamily.SelectedValue, Mode=TwoWay}" t:ScreenTip.Title="Font" t:RadComboBoxExtensions.UIInteractionCommand="{Binding SetFontFamily.Command}" t:RadComboBoxExtensions.UIInteractionCommandParameter="{Binding SelectedItem, RelativeSource={RelativeSource Self}}" Width="152">
<t:RadRibbonComboBox.ItemsPanel>
<ItemsPanelTemplate>
<VirtualizingStackPanel Width="210"/>
```

```
</ItemsPanelTemplate>
</t:RadRibbonComboBox.ItemsPanel>
<t:RadRibbonComboBox.ItemTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<TextBlock FontSize="12" FontFamily="{Binding FontFamily}" Text="{Binding
FontFamily.Source}"/>
<TextBlock Grid.Column="1" FontSize="12" FontFamily="{Binding FontFamily}" Text="{Binding
ThemeFontType, Converter={StaticResource themeFontTypeToTextConverter}}"/>
</Grid>
</DataTemplate>
</t:RadRibbonComboBox.ItemTemplate>
</t:RadRibbonComboBox>
<t:RadRibbonComboBox CanAutocompleteSelectItems="False"
CanKeyboardNavigationSelectItems="False" t:ScreenTip.Description="Change the size of your
text." IsTabStop="False" IsEditable="False" ItemsSource="{Binding FontsProvider.FontSizes,
ElementName=radSpreadsheet}" IsEnabled="{Binding SetFontSize.IsEnabled}" Margin="-1 0 0 0"
MaxDropDownHeight="400" SelectedValue="{Binding SetFontSize.SelectedValue,
ConverterParameter=Point, Converter={StaticResource dipToUnitConverter}, Mode=TwoWay}"
t:ScreenTip.Title="Font Size" t:RadComboBoxExtensions.UIInteractionCommand="{Binding
SetFontSize.Command}" t:RadComboBoxExtensions.UIInteractionCommandParameter="{Binding
SelectedItem, ConverterParameter=Point, Converter={StaticResource unitToDipConverter},
RelativeSource={RelativeSource Self}}" Width="62"/>
</StackPanel>
<t:RadButtonGroup>
<t:RadRibbonButton CommandParameter="{Binding IncreaseFontSize.SelectedValue}"
Command="{Binding IncreaseFontSize.Command}" t:ScreenTip.Description="Increase the font
size." IsEnabled="{Binding IncreaseFontSize.IsEnabled}" Size="Small"
SmallImage="{t:IconResource IconRelativePath=16/fontSizeIncrease.png,
IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Increase Font Size"/>
<t:RadRibbonButton CommandParameter="{Binding DecreaseFontSize.SelectedValue}"
Command="{Binding DecreaseFontSize.Command}" t:ScreenTip.Description="Decrease the font
size." IsEnabled="{Binding DecreaseFontSize.IsEnabled}" Size="Small"
SmallImage="{t:IconResource IconRelativePath=16/fontSizeDecrease.png,
IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Decrease Font Size"/>
</t:RadButtonGroup>
<t:RadButtonGroup x:Name="FontStylesGroup">
<t:RadButtonGroup.Resources>
```

```
<Controls:UnderlineTypeToBooleanConverter x:Key="underlineTypeToBooleanConverter"/>
</t:RadButtonGroup.Resources>
<t:RadRibbonToggleButton CommandParameter="{Binding SetIsBold.SelectedValue}"
Command="{Binding SetIsBold.Command}" t:ScreenTip.Description="Make the selected text
bold." IsChecked="{Binding SetIsBold.SelectedValue, Mode=TwoWay}" IsEnabled="{Binding
SetIsBold.IsEnabled}" Size="Small" SmallImage="{t:IconResource IconRelativePath=16/bold.png,
IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Bold (Ctrl+B)"/>
<t:RadRibbonToggleButton CommandParameter="{Binding SetIsItalic.SelectedValue}"
Command="{Binding SetIsItalic.Command}" t:ScreenTip.Description="Italicize the selected text."
IsChecked="{Binding SetIsItalic.SelectedValue, Mode=TwoWay}" IsEnabled="{Binding
SetIsItalic.IsEnabled}" Size="Small" SmallImage="{t:IconResource IconRelativePath=16/italic.png,
IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Italic (Ctrl+I)"/>
<t:RadRibbonToggleButton CommandParameter="{Binding SetUnderline.SelectedValue}"
Command="{Binding SetUnderline.Command}" t:ScreenTip.Description="Underline the selected
text." IsChecked="{Binding SetUnderline.SelectedValue, ConverterParameter=Single,
Converter={StaticResource underlineTypeToBooleanConverter}, Mode=TwoWay}"
IsEnabled="{Binding SetUnderline.IsEnabled}" Size="Small" SmallImage="{t:IconResource
IconRelativePath=16/underline.png, IconSources={StaticResource IconPaths}}"
t:ScreenTip.Title="Underline (Ctrl+U)"/>
</t:RadButtonGroup>
<t:RadButtonGroup>
<Controls:BordersMenu RadSpreadsheet="{Binding ElementName=radSpreadsheet,
Mode=OneTime}" SelectedItemImage="{t:IconResource IconRelativePath=16/bottomBorder.png,
IconSources={StaticResource IconPaths}}"/>
</t:RadButtonGroup>
<t:RadButtonGroup>
<t:RadColorPicker AutomaticColor="Transparent" BorderThickness="0"
CommandParameter="{Binding SetFillColor.SelectedValue}" Command="{Binding
SetFillColor.Command}" t:ScreenTip.Description="Color the background of selected cells."
HeaderPaletteItemsSource="{Binding ColorPalette.HeaderPaletteItemsSource,
ElementName=radSpreadsheet}" IsRecentColorsActive="True" IsEnabled="{Binding
SetFillColor.IsEnabled}" MainPaletteOrientation="{Binding ColorPalette.MainPaletteOrientation,
ElementName=radSpreadsheet}" MainPaletteItemsSource="{Binding
ColorPalette.MainPaletteItemsSource, ElementName=radSpreadsheet}" NoColorText="No Fill"
RecentColorsVisibility="Visible" SelectedColor="{Binding SetFillColor.SelectedValue,
Mode=TwoWay}" t:ScreenTip.Title="Fill Color">
<t:RadColorPicker.AdditionalContent>
<t:RadButton BorderThickness="0" Background="Transparent" CommandParameter="{Binding
SetFillColor}" Command="{Binding ShowMoreColorsDialog.Command}"
HorizontalContentAlignment="Left" IsEnabled="{Binding ShowMoreColorsDialog.IsEnabled}"
Margin="0 -1 0 0" Controls:RadControlsExtensions.ShouldCloseParentPopupOnClick="True">
<StackPanel Orientation="Horizontal">
```

```
<Image Height="16" Source="{t:IconResource IconRelativePath=16/color.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
<TextBlock FontWeight="Normal" Margin="10 0 0 0" Text="More Colors..."/>
</StackPanel>
</t:RadButton>
</t:RadColorPicker.AdditionalContent>
<t:RadColorPicker.ContentTemplate>
<DataTemplate>
<Grid>
<Image Source="{t:IconResource IconRelativePath=16/bucket.png, IconSources={StaticResource
IconPaths}}" Stretch="None" Width="16"/>
<Rectangle HorizontalAlignment="Stretch" Height="4" VerticalAlignment="Bottom">
<Rectangle.Fill>
<SolidColorBrush Color="{Binding}"/>
</Rectangle.Fill>
</Rectangle>
</Grid>
</DataTemplate>
</t:RadColorPicker.ContentTemplate>
</t:RadColorPicker>
<t:RadColorPicker BorderThickness="0" CommandParameter="{Binding
SetForeColor.SelectedValue}" Command="{Binding SetForeColor.Command}"
t:ScreenTip.Description="Change the text color." HeaderPaletteItemsSource="{Binding
ColorPalette.HeaderPaletteItemsSource, ElementName=radSpreadsheet}" IsEnabled="{Binding
SetForeColor.IsEnabled}" MainPaletteOrientation="{Binding ColorPalette.MainPaletteOrientation,
ElementName=radSpreadsheet}" MainPaletteItemsSource="{Binding
ColorPalette.MainPaletteItemsSource, ElementName=radSpreadsheet}" NoColorText="Automatic"
SelectedColor="{Binding SetForeColor.SelectedValue, Mode=TwoWay}" t:ScreenTip.Title="Text
Highlight Color">
<t:RadColorPicker.AdditionalContent>
<t:RadButton BorderThickness="0" Background="Transparent" CommandParameter="{Binding
SetForeColor}" Command="{Binding ShowMoreColorsDialog.Command}"
HorizontalContentAlignment="Left" IsEnabled="{Binding ShowMoreColorsDialog.IsEnabled}"
Margin="0 -1 0 0" Controls:RadControlsExtensions.ShouldCloseParentPopupOnClick="True">
<StackPanel Orientation="Horizontal">
<Image Height="16" Source="{t:IconResource IconRelativePath=16/color.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
<TextBlock FontWeight="Normal" Margin="10 0 0 0" Text="More Colors..."/>
</StackPanel>
</t:RadButton>
</t:RadColorPicker.AdditionalContent>
<t:RadColorPicker.ContentTemplate>
```

```
<DataTemplate>
<Grid>
<Image Source="{t:IconResource IconRelativePath=16/foregroundColor.png,
IconSources={StaticResource IconPaths}}" Stretch="None" Width="16"/>
<Rectangle HorizontalAlignment="Stretch" Height="3" VerticalAlignment="Bottom">
<Rectangle.Fill>
<SolidColorBrush Color="{Binding}"/>
</Rectangle.Fill>
</Rectangle>
</Grid>
</DataTemplate>
</t:RadColorPicker.ContentTemplate>
</t:RadColorPicker>
</t:RadButtonGroup>
</t:RadOrderedWrapPanel>
</t:RadRibbonGroup>
<t:RadRibbonGroup DialogLauncherCommandParameter="Alignment"
t:ScreenTip.Description="Show the Alignment tab of the Format Cells dialog box."
DialogLauncherCommand="{Binding ShowFormatCellsDialog.Command}" Header="Alignment"
IsEnabled="{Binding AlignmentGroup.IsEnabled}" t:ScreenTip.Title="Format Cells: Alignment">
<t:RadRibbonGroup.Resources>
<Controls:RadHorizontalAlignmentToBooleanConverter
x:Key="horizontalAlignmentToBooleanConverter"/>
<Controls:RadVerticalAlignmentToBooleanConverter
x:Key="verticalAlignmentToBooleanConverter"/>
</t:RadRibbonGroup.Resources>
<t:RadRibbonGroup.DialogLauncherVisibility>
<Binding Converter="{StaticResource BoolToVisibilityValueConverter}"
Path="ShowFormatCellsDialog.IsEnabled"/>
</t:RadRibbonGroup.DialogLauncherVisibility>
<t:RadRibbonGroup.Variants>
<t:GroupVariant Priority="0" Variant="Large"/>
</t:RadRibbonGroup.Variants>
<t:RadOrderedWrapPanel>
<t:RadButtonGroup>
<t:RadRibbonToggleButton CommandParameter="{Binding SetVerticalAlignment.SelectedValue}"
Command="{Binding SetVerticalAlignment.Command}" t:ScreenTip.Description="Align the text to
the top of the cell." IsChecked="{Binding SetVerticalAlignment.SelectedValue,
ConverterParameter=Top, Converter={StaticResource verticalAlignmentToBooleanConverter},
Mode=TwoWay}" IsEnabled="{Binding SetVerticalAlignment.IsEnabled}" Size="Small"
SmallImage="{t:IconResource IconRelativePath=16/alignTop.png, IconSources={StaticResource
IconPaths}}" t:ScreenTip.Title="Top Align"/>
```

```
<t:RadRibbonToggleButton CommandParameter="{Binding SetVerticalAlignment.SelectedValue}"
Command="{Binding SetVerticalAlignment.Command}" t:ScreenTip.Description="Align text so that
it is centered between the top and the bottom of the cell." IsChecked="{Binding
SetVerticalAlignment.SelectedValue, ConverterParameter=Center, Converter={StaticResource
verticalAlignmentToBooleanConverter}, Mode=TwoWay}" IsEnabled="{Binding
SetVerticalAlignment.IsEnabled}" Size="Small" SmallImage="{t:IconResource
IconRelativePath=16/alignMiddle.png, IconSources={StaticResource IconPaths}}"
t:ScreenTip.Title="Middle Align"/>
<t:RadRibbonToggleButton CommandParameter="{Binding SetVerticalAlignment.SelectedValue}"
Command="{Binding SetVerticalAlignment.Command}" t:ScreenTip.Description="Align the text to
the bottom of the cell." IsChecked="{Binding SetVerticalAlignment.SelectedValue,
ConverterParameter=Bottom, Converter={StaticResource verticalAlignmentToBooleanConverter},
Mode=TwoWay}" IsEnabled="{Binding SetVerticalAlignment.IsEnabled}" Size="Small"
SmallImage="{t:IconResource IconRelativePath=16/alignBottom.png,
IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Bottom Align"/>
</t:RadButtonGroup>
<t:RadButtonGroup>
<t:RadRibbonToggleButton CommandParameter="{Binding
SetHorizontalAlignment.SelectedValue}" Command="{Binding
SetHorizontalAlignment.Command}" t:ScreenTip.Description="Align text to the left."
IsChecked="{Binding SetHorizontalAlignment.SelectedValue, ConverterParameter=Left,
Converter={StaticResource horizontalAlignmentToBooleanConverter}, Mode=TwoWay}"
IsEnabled="{Binding SetHorizontalAlignment.IsEnabled}" Size="Small"
SmallImage="{t:IconResource IconRelativePath=16/alignLeft.png, IconSources={StaticResource
IconPaths}}" t:ScreenTip.Title="Align Text Left"/>
<t:RadRibbonToggleButton CommandParameter="{Binding
SetHorizontalAlignment.SelectedValue}" Command="{Binding
SetHorizontalAlignment.Command}" t:ScreenTip.Description="Center text." IsChecked="{Binding
SetHorizontalAlignment.SelectedValue, ConverterParameter=Center, Converter={StaticResource
horizontalAlignmentToBooleanConverter}, Mode=TwoWay}" IsEnabled="{Binding
SetHorizontalAlignment.IsEnabled}" Size="Small" SmallImage="{t:IconResource
IconRelativePath=16/alignCenter.png, IconSources={StaticResource IconPaths}}"
t:ScreenTip.Title="Center"/>
<t:RadRibbonToggleButton CommandParameter="{Binding
SetHorizontalAlignment.SelectedValue}" Command="{Binding
SetHorizontalAlignment.Command}" t:ScreenTip.Description="Align text to the right."
IsChecked="{Binding SetHorizontalAlignment.SelectedValue, ConverterParameter=Right,
Converter={StaticResource horizontalAlignmentToBooleanConverter}, Mode=TwoWay}"
IsEnabled="{Binding SetHorizontalAlignment.IsEnabled}" Size="Small"
SmallImage="{t:IconResource IconRelativePath=16/alignRight.png, IconSources={StaticResource
IconPaths}}" t:ScreenTip.Title="Align Text Right"/>
</t:RadButtonGroup>
```

<t:RadButtonGroup>
<t:RadRibbonButton CommandParameter="{Binding DecreaseIndent.SelectedValue}"
Command="{Binding DecreaseIndent.Command}" t:ScreenTip.Description="Decrease the margin
between the border and the text in the cell." IsEnabled="{Binding DecreaseIndent.IsEnabled}"
Size="Small" SmallImage="{t:IconResource IconRelativePath=16/indentDecrease.png,
IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Decrease Indent"/>
<t:RadRibbonButton CommandParameter="{Binding IncreaseIndent.SelectedValue}"
Command="{Binding IncreaseIndent.Command}" t:ScreenTip.Description="Increase the margin
between the border and the text in the cell." IsEnabled="{Binding IncreaseIndent.IsEnabled}"
Size="Small" SmallImage="{t:IconResource IconRelativePath=16/indentIncrease.png,
IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Increase Indent"/>
</t:RadButtonGroup>
</t:RadOrderedWrapPanel>
<t:Separator/>
<t:RadOrderedWrapPanel>
<t:RadOrderedWrapPanel.Resources>
<t:InvertedBooleanConverter x:Key="InvertedBooleanConverter"/>
</t:RadOrderedWrapPanel.Resources>
<t:RadRibbonToggleButton CommandParameter="{Binding SetIsWrapped.SelectedValue}"
Command="{Binding SetIsWrapped.Command}" t:ScreenTip.Description="Make all content visible
within a cell by displaying it on multiple lines." IsChecked="{Binding SetIsWrapped.SelectedValue,
Mode=TwoWay}" IsEnabled="{Binding SetIsWrapped.IsEnabled}" Size="Medium"
SmallImage="{t:IconResource IconRelativePath=16/wrapText.png, IconSources={StaticResource
IconPaths}}" Text="Wrap Text" t:ScreenTip.Title="Wrap Text"/>
<t:RadRibbonSplitButton x:Name="MergeAndCenterButton" CommandParameter="{Binding
MergeAndCenter.SelectedValue, Converter={StaticResource InvertedBooleanConverter}}"
Command="{Binding MergeAndCenter.Command}" t:ScreenTip.Description="Joins the selected
cells into one larger cell and centers the contents in the new cell.

This is often used to create labels that span multiple columns." IsToggle="True"
IsChecked="{Binding MergeAndCenter.SelectedValue, Mode=TwoWay}" IsEnabled="{Binding
MergeAndCenter.IsEnabled}" Size="Medium" SmallImage="{t:IconResource
IconRelativePath=16/mergeAndCenter.png, IconSources={StaticResource IconPaths}}"
Text="Merge & Center" t:ScreenTip.Title="Merge & Center">
<t:RadRibbonSplitButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuItem CommandParameter="{Binding MergeAndCenter.SelectedValue}"
Command="{Binding MergeAndCenter.Command}" t:ScreenTip.Description="Joins the selected
cells into one larger cell and centers the contents in the new cell.

This is often used to create labels that span multiple columns." Header="Merge & Center"
IsChecked="{Binding MergeAndCenter.SelectedValue, Mode=TwoWay}" IsEnabled="{Binding
MergeAndCenter.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Merge & Center">
<t:RadMenuItem.Icon>

```
<Image Source="{t:IconResource IconRelativePath=16/mergeAndCenter.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding MergeAcross.Command}" t:ScreenTip.Description="Merge
each row of the selected cells into a larger cell." Header="Merge Across" IsEnabled="{Binding
MergeAcross.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Merge Across">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/mergeAcross.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding Merge.Command}" t:ScreenTip.Description="Merge the
selected cells into one cell." Header="Merge Cells" IsEnabled="{Binding Merge.IsEnabled,
Mode=TwoWay}" t:ScreenTip.Title="Merge Cells">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/merge.png, IconSources={StaticResource
IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding Unmerge.Command}" t:ScreenTip.Description="Split the
selected cells into new multiple cells." Header="Unmerge Cells" IsEnabled="{Binding
Unmerge.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Unmerge Cells">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/unmerge.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadContextMenu>
</t:RadRibbonSplitButton.DropDownContent>
</t:RadRibbonSplitButton>
</t:RadOrderedWrapPanel>
</t:RadRibbonGroup>
<t:RadRibbonGroup DialogLauncherVisibility="{Binding ShowFormatCellsDialog.IsEnabled,
Converter={StaticResource BoolToVisibilityValueConverter}}"
DialogLauncherCommandParameter="Number" t:ScreenTip.Description="Show the Number tab of
the Format Cells dialog box." DialogLauncherCommand="{Binding
ShowFormatCellsDialog.Command}" Header="Number" IsEnabled="{Binding
NumberGroup.IsEnabled}" t:ScreenTip.Title="Format Cells: Number">
<t:RadRibbonGroup.Variants>
<t:GroupVariant Priority="0" Variant="Large"/>
</t:RadRibbonGroup.Variants>
```

<t:RadOrderedWrapPanel>
<Controls:NumberFormatAutoCompleteComboBox CellValueFormat="{Binding CommandDescriptors.SetFormat.SelectedValue, ElementName=radSpreadsheet, Mode=TwoWay}" t:ScreenTip.Description="Choose how the values in a cell are displayed: as a percentage, as currency, as a date or time, etc." IsEnabled="{Binding CommandDescriptors.SetFormat.IsEnabled, ElementName=radSpreadsheet, Mode=TwoWay}" RadSpreadsheet="{Binding ElementName=radSpreadsheet, Mode=OneTime}" t:ScreenTip.Title="Number Format"/>
<t:RadButtonGroup Margin="0 0 3 0">
<t:RadRibbonSplitButton x:Name="AccountingNumberFormatButton" CommandParameter="Currency" Command="{Binding SetStyle.Command}" t:ScreenTip.Description="Choose an alternate currency format for the selected cell.

For instance, choose Euros instead of Dollars." IsEnabled="{Binding SetStyle.IsEnabled}" Size="Small" SmallImage="{t:IconResource IconRelativePath=16/accountingNumberFormat.png, IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Accounting Number Format">
<t:RadRibbonSplitButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuItem CommandParameter="_([$$-409]* #,##0.00_);_([$$-409]* (#,##0.00);_([$$-409]* "-"??_);_(@_)" Command="{Binding SetFormat.Command}" Header="$ English (U.S.)" IsEnabled="{Binding SetFormat.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuItem CommandParameter="_-[$£-809]* #,##0.00_-;-[$£-809]* #,##0.00_-;_-[$£-809]* "-"??_-;_-@_-" Command="{Binding SetFormat.Command}" Header="£ English (U.K.)" IsEnabled="{Binding SetFormat.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuItem CommandParameter="_([$€-2] * #,##0.00_);_([$€-2] * (#,##0.00);_([$€-2] * "-"??_);_(@_)" Command="{Binding SetFormat.Command}" Header="€ Euro (€ 123)" IsEnabled="{Binding SetFormat.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuItem CommandParameter="_ [$¥-804]* #,##0.00_ ;_ [$¥-804]* -#,##0.00_ ;_ [$¥-804]* "-"??_ ;_ @_ " Command="{Binding SetFormat.Command}" Header="¥ Chinese (PRC)" IsEnabled="{Binding SetFormat.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuItem CommandParameter="_ [$fr.-100C] * #,##0.00_ ;_ [$fr.-100C] * -#,##0.00_ ;_ [$fr.-100C] * "-"??_ ;_ @_ " Command="{Binding SetFormat.Command}" Header="fr. French (Switzerland)" IsEnabled="{Binding SetFormat.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem CommandParameter="_($* #,##0.00_);_($* (#,##0.00);_($* "-"??_);_(@_)" Command="{Binding ShowFormatCellsDialogNumberTab.Command}" Header="More Accounting Formats..." IsEnabled="{Binding ShowFormatCellsDialogNumberTab.IsEnabled, Mode=TwoWay}"/>
</t:RadContextMenu>
</t:RadRibbonSplitButton.DropDownContent>
</t:RadRibbonSplitButton>
<t:RadRibbonButton CommandParameter="Percent" Command="{Binding SetStyle.Command}"

t:ScreenTip.Description="Display the value of the cell as a percentage." IsEnabled="{Binding SetStyle.IsEnabled}" Size="Small" SmallImage="{t:IconResource IconRelativePath=16/percent.png, IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Percent Style (Ctrl+Shift+%)"/>
<t:RadRibbonButton CommandParameter="Comma" Command="{Binding SetStyle.Command}" t:ScreenTip.Description="Display the value of the cell with a thousands separator.

This will change the format of the cell to Accounting without a currency symbol." IsEnabled="{Binding SetStyle.IsEnabled}" Size="Small" SmallImage="{t:IconResource IconRelativePath=16/edit-comma.png, IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Comma Style"/>
</t:RadButtonGroup>
<t:RadButtonGroup>
<t:RadRibbonButton Command="{Binding IncreaseDecimalPlaces.Command}" t:ScreenTip.Description="Show more precise values by showing more decimal places." IsEnabled="{Binding IncreaseDecimalPlaces.IsEnabled}" Size="Small" SmallImage="{t:IconResource IconRelativePath=16/decimal.png, IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Increase Decimal"/>
<t:RadRibbonButton Command="{Binding DecreaseDecimalPlaces.Command}" t:ScreenTip.Description="Show less precise values by showing fewer decimal places." IsEnabled="{Binding DecreaseDecimalPlaces.IsEnabled}" Size="Small" SmallImage="{t:IconResource IconRelativePath=16/decimalDecrease.png, IconSources={StaticResource IconPaths}}" t:ScreenTip.Title="Decrease Decimal"/>
</t:RadButtonGroup>
</t:RadOrderedWrapPanel>
</t:RadRibbonGroup>
<t:RadRibbonGroup Header="Cells" IsEnabled="{Binding CellsGroup.IsEnabled}">
<t:RadRibbonGroup.Variants>
<t:GroupVariant Priority="0" Variant="Large"/>
</t:RadRibbonGroup.Variants>
<t:RadRibbonDropDownButton x:Name="InsertCellsButton" t:ScreenTip.Description="Insert cells, rows, or columns into the sheet or table." IsEnabled="{Binding CellsGroupInsert.IsEnabled}" LargeImage="{t:IconResource IconRelativePath=32/cellsInsert.png, IconSources={StaticResource IconPaths}}" Size="Large" Text="Insert" t:ScreenTip.Title="Insert Cells">
<t:RadRibbonDropDownButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuItem Command="{Binding InsertCells.Command}" t:ScreenTip.Description="Add new rows, columns, cells or sheets to your workbook.

New rows will be added above the selection and new columns will be added to the left of the selection.
 Add three columns at once by selecting three existing columns first; this also works with rows." Header="Insert Cells..." IsEnabled="{Binding InsertCells.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Insert Cells">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/cellsInsert.png,

```
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Command="{Binding InsertRows.Command}" Header="Insert Sheet Rows"
IsEnabled="{Binding InsertRows.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/rowsInsert.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding InsertColumns.Command}" Header="Insert Sheet
Columns" IsEnabled="{Binding InsertColumns.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/columnsInsert.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem CommandParameter="Worksheet" Command="{Binding InsertSheet.Command}"
Header="Insert Sheet" IsEnabled="{Binding InsertSheet.IsEnabled, Mode=TwoWay}"
ToolTipService.ToolTip="Insert Worksheet (F11)">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/sheetInsert.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadContextMenu>
</t:RadRibbonDropDownButton.DropDownContent>
</t:RadRibbonDropDownButton>
<t:RadRibbonDropDownButton x:Name="DeleteCellsButton" t:ScreenTip.Description="Delete
cells, rows, or columns from the sheet or table." IsEnabled="{Binding
CellsGroupDelete.IsEnabled}" LargeImage="{t:IconResource
IconRelativePath=32/cellsRemove.png, IconSources={StaticResource IconPaths}}" Size="Large"
Text="Delete" t:ScreenTip.Title="Delete Cells">
<t:RadRibbonDropDownButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuItem Command="{Binding RemoveCells.Command}" t:ScreenTip.Description="Delete
rows, columns or cells." Header="Delete Cells..." IsEnabled="{Binding RemoveCells.IsEnabled,
Mode=TwoWay}" t:ScreenTip.Title="Delete Cells">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/cellsRemove.png,
```

```
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Command="{Binding RemoveRows.Command}" Header="Delete Sheet Rows"
IsEnabled="{Binding RemoveRows.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/rowsRemove.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding RemoveColumns.Command}" Header="Delete Sheet
Columns" IsEnabled="{Binding RemoveColumns.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/columnsRemove.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Command="{Binding RemoveSheet.Command}" Header="Delete Sheet"
IsEnabled="{Binding RemoveSheet.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/sheetRemove.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadContextMenu>
</t:RadRibbonDropDownButton.DropDownContent>
</t:RadRibbonDropDownButton>
<t:RadRibbonDropDownButton x:Name="FormatButton" t:ScreenTip.Description="Change the row
height or column width, organize sheets, or protect or hide cells." IsEnabled="{Binding
CellsGroupFormat.IsEnabled}" LargeImage="{t:IconResource
IconRelativePath=32/cellsFormat.png, IconSources={StaticResource IconPaths}}" Size="Large"
Text="Format" t:ScreenTip.Title="Format">
<t:RadRibbonDropDownButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuGroupItem BorderThickness="0" Background="Transparent" Header="Cells Size"
Margin="-1">
<t:RadMenuItem Command="{Binding SetRowsHeight.Command}" Header="Row Height..."
IsEnabled="{Binding SetRowsHeight.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/rowHeight.png,
```

```
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding AutoFitRowsHeight.Command}" Header="AutoFit Row
Height" IsEnabled="{Binding AutoFitRowsHeight.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuItem Command="{Binding SetDefaultRowHeight.Command}" Header="Default
Height..." IsEnabled="{Binding SetDefaultRowHeight.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Command="{Binding SetColumnsWidth.Command}" Header="Column Width..."
IsEnabled="{Binding SetColumnsWidth.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/columnWidth.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding AutoFitColumnsWidth.Command}" Header="AutoFit
Column Width" IsEnabled="{Binding AutoFitColumnsWidth.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuItem Command="{Binding SetDefaultColumnWidth.Command}" Header="Default
Width..." IsEnabled="{Binding SetDefaultColumnWidth.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Command="{Binding HideRows.Command}" Header="Hide Rows"
IsEnabled="{Binding HideRows.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuItem Command="{Binding HideColumns.Command}" Header="Hide Columns"
IsEnabled="{Binding HideColumns.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuItem Command="{Binding UnhideRows.Command}" Header="Unhide Rows"
IsEnabled="{Binding UnhideRows.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuItem Command="{Binding UnhideColumns.Command}" Header="Unhide Columns"
IsEnabled="{Binding UnhideColumns.IsEnabled, Mode=TwoWay}"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Command="{Binding ShowFormatCellsDialog.Command}"
t:ScreenTip.Description="Datasheet formatting" Header="Format Cells..." IsEnabled="{Binding
ShowFormatCellsDialog.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/formatCells.png,
IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadMenuGroupItem>
</t:RadContextMenu>
</t:RadRibbonDropDownButton.DropDownContent>
</t:RadRibbonDropDownButton>
</t:RadRibbonGroup>
```

<t:RadRibbonGroup Header="Editing" IsEnabled="{Binding EditingGroup.IsEnabled}">
<t:RadRibbonGroup.Variants>
<t:GroupVariant Priority="0" Variant="Large"/>
</t:RadRibbonGroup.Variants>
<t:RadOrderedWrapPanel>
<t:RadRibbonDropDownButton x:Name="FillButton" t:ScreenTip.Description="Continue a pattern into one or more adjacent cells.

You can fill cells in any direction and into any range of adjacent cells." Size="Medium" SmallImage="{t:IconResource IconRelativePath=16/fillDown.png, IconSources={StaticResource IconPaths}}" Text="Fill" t:ScreenTip.Title="Fill">
<t:RadRibbonDropDownButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuItem Command="{Binding FillDown.Command}" Header="Down" IsEnabled="{Binding FillDown.IsEnabled, Mode=TwoWay}" ToolTipService.ToolTip="Fill Down (Ctrl+D)">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/fillDown.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding FillRight.Command}" Header="Right" IsEnabled="{Binding FillRight.IsEnabled, Mode=TwoWay}" ToolTipService.ToolTip="Fill Right (Ctrl+R)">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/fillRight.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding FillUp.Command}" Header="Up" IsEnabled="{Binding FillUp.IsEnabled, Mode=TwoWay}" ToolTipService.ToolTip="Fill Up">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/fillUp.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding FillLeft.Command}" Header="Left" IsEnabled="{Binding FillLeft.IsEnabled, Mode=TwoWay}" ToolTipService.ToolTip="Fill Left">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/fillLeft.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding ShowSeriesDialog.Command}" Header="Series..." IsEnabled="{Binding ShowSeriesDialog.IsEnabled, Mode=TwoWay}" ToolTipService.ToolTip="Fill Series"/>

```
</t:RadContextMenu>
</t:RadRibbonDropDownButton.DropDownContent>
</t:RadRibbonDropDownButton>
<t:RadRibbonDropDownButton x:Name="ClearButton" t:ScreenTip.Description="Delete everything from the cell, or selectively remove the formatting, the contents, or the comments." Size="Medium" SmallImage="{t:IconResource IconRelativePath=16/clear.png, IconSources={StaticResource IconPaths}}" Text="Clear" t:ScreenTip.Title="Clear">
<t:RadRibbonDropDownButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuItem Command="{Binding Clear.Command}" t:ScreenTip.Description="Clear everything from the selected cells. &amp;#xA;&amp;#xA;All contents, formatting, and comments are cleared from the selected cells." Header="Clear All" IsEnabled="{Binding Clear.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Clear All">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/clear.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem CommandParameter="Formats" Command="{Binding Clear.Command}" t:ScreenTip.Description="Clear only the formatting that is applied to the selected cells." Header="Clear Formats" IsEnabled="{Binding Clear.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Clear Formats">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/clearFormats.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem CommandParameter="Contents" Command="{Binding Clear.Command}" t:ScreenTip.Description="Clear only the contents in the selected cells. &amp;#xA;&amp;#xA;The formatting and comments are not cleared." Header="Clear Contents" IsEnabled="{Binding Clear.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Clear Contents (Del)"/>
<t:RadMenuItem CommandParameter="Hyperlinks" Command="{Binding Clear.Command}" t:ScreenTip.Description="Clear the hyperlinks from the selected cells." Header="Clear Hyperlinks" IsEnabled="{Binding Clear.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Clear Hyperlinks"/>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Command="{Binding RemoveHyperlink.Command}" Header="Remove Hyperlink" IsEnabled="{Binding RemoveHyperlink.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/removeHyperlink.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
```

```
</t:RadContextMenu>
</t:RadRibbonDropDownButton.DropDownContent>
</t:RadRibbonDropDownButton>
</t:RadOrderedWrapPanel>
<t:RadRibbonDropDownButton t:ScreenTip.Description="Organize your data so it's easier to analize. You can sort the selected data from smallest to largest, largest to smallest or filter out specific values." IsEnabled="{Binding SortAndFilterGroup.IsEnabled}" LargeImage="{t:IconResource IconRelativePath=32/sortAndFilter.png, IconSources={StaticResource IconPaths}}" Size="Large" Text="Sort &amp;amp; Filter" t:ScreenTip.Title="Sort &amp;amp; Filter">
<t:RadRibbonDropDownButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuGroupItem BorderThickness="0" Background="Transparent" Margin="-1">
<t:RadMenuItem CommandParameter="Ascending" Command="{Binding Sort.Command}" t:ScreenTip.Description="Lowest to highest." Header="Sort A to Z" IsEnabled="{Binding Sort.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Sort A to Z">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/sortAscending.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem CommandParameter="Descending" Command="{Binding Sort.Command}" t:ScreenTip.Description="Highest to lowest." Header="Sort Z to A" IsEnabled="{Binding Sort.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Sort Z to A">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/sortDescending.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding ShowSortingDialog.Command}" t:ScreenTip.Description="Shows the Sort dialog box to sort data based on several criteria at once." Header="Custom sort..." IsEnabled="{Binding ShowSortingDialog.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Sort">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/customSort.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuSeparatorItem/>
<t:RadMenuItem Command="{Binding SetFilterRange.Command}" t:ScreenTip.Description="Turn on filtering for the selected cells. Then, click the arrow in the column header to narrow down the data." Header="Filter" IsEnabled="{Binding SetFilterRange.IsEnabled}" t:ScreenTip.Title="Filter">
```

```
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/filter.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding ClearFilters.Command}" t:ScreenTip.Description="Clear the filter for the current range of data." Header="Clear" IsEnabled="{Binding ClearFilters.IsEnabled}" t:ScreenTip.Title="Clear">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/filterClear.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding ReapplyFilters.Command}" t:ScreenTip.Description="Reapply the filter on the current range so that changes you've made are included." Header="Reapply" IsEnabled="{Binding ReapplyFilters.IsEnabled}" t:ScreenTip.Title="Reapply">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/filterReapply.png, IconSources={StaticResource IconPaths}}" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadMenuGroupItem>
</t:RadContextMenu>
</t:RadRibbonDropDownButton.DropDownContent>
</t:RadRibbonDropDownButton>
<t:RadRibbonButton Content="Find" CommandParameter="Find" Command="{Binding ShowFindAndReplaceDialog.Command}" t:ScreenTip.Description="Use advanced search options to replace text or pick other ways to narrow your search." LargeImage="{t:IconResource IconRelativePath=32/findAndSelect.png, IconSources={StaticResource IconPaths}}" Size="Large" t:ScreenTip.Title="Find"/>
</t:RadRibbonGroup>
</t:RadRibbonTab>
<t:RadRibbonTab Header="Test Info" DataContext="{Binding ElementName=DataFormattingToolControl, Path=DataContext}">
<t:RadRibbonGroup Header="Test" >
<t:RadRibbonButton x:Name="Test_Numerator_Btn" Text="Test Number" VerticalContentAlignment="Center"
Command="{Binding Path=NumerateTestsCommand}" t:ScreenTip.Description="Tags the selected test number column." t:ScreenTip.Title="Tag Test Number"
/>
<t:Separator/>
```

<t:RadRibbonButton x:Name="Test_Name_Btn" Text="Test Name"
VerticalContentAlignment="Center"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Test Name" t:ScreenTip.Description="Tags the selected test number column." t:ScreenTip.Title="Tag Test Number"/>
<t:Separator/>
<t:RadRibbonButton x:Name="Test_Unit_Btn" Text="Test Unit"
VerticalContentAlignment="Center"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Test Unit" t:ScreenTip.Description="Tags the selected test unit column."
t:ScreenTip.Title="Tag Test Number"/>
</t:RadRibbonGroup>
<t:RadRibbonGroup Header="Other Info">
<t:RadRibbonButton x:Name="Group_Btn" Text="Group" VerticalContentAlignment="Center"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Group" t:ScreenTip.Description="Tags the selected group column."
t:ScreenTip.Title="Tag Group"/>
<t:Separator/>
<t:RadRibbonButton x:Name="FunctionCall_Btn" Text="Function Call"
VerticalContentAlignment="Center"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Function Call" t:ScreenTip.Description="Tags the selected Function Call column." t:ScreenTip.Title="Tag Function Call"/>
</t:RadRibbonGroup>
<t:RadRibbonGroup Header="Parameters">
<t:RadRibbonButton x:Name="Parameter_Btn" Text="Parameter"
VerticalContentAlignment="Center"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Parameter" t:ScreenTip.Description="Tags the selected parameter column." t:ScreenTip.Title="Tag Parameter"/>
<t:Separator/>
<t:RadRibbonButton x:Name="Pin_Parameter_Btn" Text="Pin Parameter"
VerticalContentAlignment="Center"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Pin Parameter" t:ScreenTip.Description="Tags the selected pin parameter column." t:ScreenTip.Title="Tag Pin Parameter"/>
</t:RadRibbonGroup>
</t:RadRibbonTab >
<t:RadRibbonTab Header="DC Conditions" DataContext="{Binding
ElementName=DataFormattingToolControl, Path=DataContext}">
<t:RadRibbonGroup Header="Modes">
<t:RadRibbonButton x:Name="Digital_Mode_Btn" Content="Digital Mode"

```
VerticalContentAlignment="Center"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Digital Mode" t:ScreenTip.Description="Tags the selected digital mode
column."  t:ScreenTip.Title="Tag Digital Mode"/>
</t:RadRibbonGroup>
</t:RadRibbonTab>
<t:RadRibbonTab Header="RF Conditions" DataContext="{Binding
ElementName=DataFormattingToolControl, Path=DataContext}">
<t:RadRibbonGroup >
<t:RadRibbonButton x:Name="Pin_Btn" Content="Pin" VerticalContentAlignment="Center"
Width="50"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Pin" t:ScreenTip.Description="Tags the selected Pin column."
t:ScreenTip.Title="Tag Pin"/>
<t:Separator/>
<t:RadRibbonButton x:Name="Pout_Btn" Content="Pout" VerticalContentAlignment="Center"
Width="50"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Pout" t:ScreenTip.Description="Tags the selected Pout column."
t:ScreenTip.Title="Tag Pout"/>
<t:Separator/>
<t:RadRibbonButton x:Name="Pout_Tolerance_Btn" Content="Pout Tolerance"
VerticalContentAlignment="Center"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Pout_Tolerance" t:ScreenTip.Description="Tags the selected Pout
Tolerance column."  t:ScreenTip.Title="Tag Pout Tolerance"/>
<t:Separator/>
<t:RadRibbonButton x:Name="Frequency_Btn" Content="Frequency"
VerticalContentAlignment="Center"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Frequency" t:ScreenTip.Description="Tags the selected Frequency
column."  t:ScreenTip.Title="Tag Frequency"/>
<t:Separator/>
<t:RadRibbonButton x:Name="Waveforms_Btn" Content="Waveform"
VerticalContentAlignment="Center"
Command="{Binding Path=TagDataCommand}"
CommandParameter="Waveform" t:ScreenTip.Description="Tags the selected Waveform
column."  t:ScreenTip.Title="Tag Waveform"/>
</t:RadRibbonGroup>
</t:RadRibbonTab>
<t:RadRibbonTab Header="Limits" DataContext="{Binding
ElementName=DataFormattingToolControl, Path=DataContext}">
```

```
<t:RadRibbonGroup Header="Limits" DefaultVariant="Small" >
<t:RadMenuItem Header="FT Upper Limit"
Command="{Binding Path=TagDataCommand}"
CommandParameter="FT Upper Limit"
t:ScreenTip.Description="Tags the selected upper limit column." t:ScreenTip.Title="Tag Upper
Limit"/>
<t:RadMenuItem Header=" FT Lower Limit"
Command="{Binding Path=TagDataCommand}"
CommandParameter="FT Lower Limit"
t:ScreenTip.Description="Tags the selected lower limit column." t:ScreenTip.Title="Tag Lower
Limit"/>
<t:RadMenuItem  Header="QA Upper Limit" t:ScreenTip.Description="Tags the selected QA upper
limit column." t:ScreenTip.Title="Tag QA Upper Limit"/>
<t:RadMenuItem  Header="QA Lower Limit" t:ScreenTip.Description="Tags the selected QA lower
limit column." t:ScreenTip.Title="Tag QA Lower Limit"/>

</t:RadRibbonGroup>
<t:RadRibbonGroup Header="Gold" DefaultVariant="Small" >
<t:RadRibbonSplitButton  IsToggle="False" Text="Gold Tests" Width="80"
t:ScreenTip.Description="" t:ScreenTip.Title="">
<t:RadRibbonSplitButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuItem  Header="Gold Retest Lower" />
<t:RadMenuItem  Header="Gold Retest Upper" />
<t:RadMenuItem  Header="Gold Retest Control" />
<t:RadMenuItem  Header="Gold Test Lower" />
<t:RadMenuItem  Header="Gold Test Upper" />
<t:RadMenuItem  Header="Gold Test Control" />
<t:RadMenuItem  Header="Gold Linear Lower" />
<t:RadMenuItem  Header="Gold Linear Upper" />
<t:RadMenuItem  Header="Gold Linear Control" />
</t:RadContextMenu>
</t:RadRibbonSplitButton.DropDownContent>
</t:RadRibbonSplitButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Header="Stop On Fail" DefaultVariant="Small" >
<t:RadMenuItem  Header="Stop On Fail" />
<t:RadMenuItem  Header="Apply After" />
</t:RadRibbonGroup>
<t:RadRibbonGroup Header="Offset" DefaultVariant="Small" >
<t:RadRibbonSplitButton  IsToggle="False" Text="Offset Sites" Width="80"
t:ScreenTip.Description="" t:ScreenTip.Title="">
```

<t:RadRibbonSplitButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuItem Header="Offset Site 1" />
<t:RadMenuItem Header="Offset Site 2" />
<t:RadMenuItem Header="Offset Site 3" />
<t:RadMenuItem Header="Offset Site 4" />
<t:RadMenuItem Header="Offset Site 5" />
<t:RadMenuItem Header="Offset Site 6" />
<t:RadMenuItem Header="Offset Site 7" />
<t:RadMenuItem Header="Offset Site 8" />
</t:RadContextMenu>
</t:RadRibbonSplitButton.DropDownContent>
</t:RadRibbonSplitButton>
</t:RadRibbonGroup>
<t:RadRibbonGroup Header="Bins" DefaultVariant="Small" >
<t:RadMenuItem Header="Hard Bin #" t:ScreenTip.Description="Tags the selected Hard Bin Number column." t:ScreenTip.Title="Tag Hard Bin Number"/>
<t:RadMenuItem Header="Hard Bin Name" t:ScreenTip.Description="Tags the selected Hard Bin Name column." t:ScreenTip.Title="Tag Hard Bin Name"/>
<t:RadMenuItem Header="Soft Bin #" t:ScreenTip.Description="Tags the selected Soft Bin Number column." t:ScreenTip.Title="Tag Soft Bin Number"/>
<t:RadMenuItem Header="Soft Bin Name" t:ScreenTip.Description="Tags the selected QA Soft Bin Name column." t:ScreenTip.Title="Tag Soft Bin Name"/>
</t:RadRibbonGroup>
<t:RadRibbonGroup Header="Fail Priority" DefaultVariant="Small" >
<t:RadMenuItem Header="Fail Priority" />
<t:RadMenuItem Header="Stop On Fail Default" />
</t:RadRibbonGroup>
</t:RadRibbonTab>
<t:RadRibbonTab Header="RFFE Pattern" DataContext="{Binding ElementName=DataFormattingToolControl, Path=DataContext}">
<t:RadRibbonGroup Header="Modes">
<t:RadRibbonButton x:Name="Mode_Name_Btn" Content="Mode Name" VerticalContentAlignment="Center" Command="{Binding Path=NumeratePatternModesCommand}" CommandParameter="PatternMode" t:ScreenTip.Description="Tags the selected mode name column." t:ScreenTip.Title="Tag Mode"/>
<t:Separator/>
<t:RadRibbonButton x:Name="Register_Btn" Content="Register" VerticalContentAlignment="Center" Command="{Binding Path=TagDataCommand}" CommandParameter="Register" t:ScreenTip.Description="Tags the selected register column.

```
Note: Remove words like 'Register'"  t:ScreenTip.Title="Tag Register"/>
</t:RadRibbonGroup>
</t:RadRibbonTab>
<t:RadRibbonTab ContextualGroupName="PictureTools" Header="Format">
<t:RadRibbonGroup Header="Arrange">
<t:RadCollapsiblePanel>
<t:RadRibbonSplitButton x:Name="BringForwardButton" Command="{Binding
BringForward.Command}" t:ScreenTip.Description="Bring the selected object forward one level so
that it's hidden behind fewer objects." IsEnabled="{Binding BringForward.IsEnabled,
Mode=TwoWay}" Size="Medium" SmallImage="{t:IconResource
IconRelativePath=16/bringForward.png, IconSources={StaticResource IconPaths}}" Text="Bring
Forward" t:ScreenTip.Title="Bring Forward">
<t:RadRibbonSplitButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuItem Command="{Binding BringForward.Command}" t:ScreenTip.Description="Bring
the selected object forward one level so that it's hidden behind fewer objects." Header="Bring
Forward" IsEnabled="{Binding BringForward.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Bring
Forward">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/bringForward.png,
IconSources={StaticResource IconPaths}}" Stretch="None" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding BringToFront.Command}" t:ScreenTip.Description="Bring
the selected object in front of all other objects." Header="Bring to Front" IsEnabled="{Binding
BringToFront.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Bring to front">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/bringToFront.png,
IconSources={StaticResource IconPaths}}" Stretch="None" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadContextMenu>
</t:RadRibbonSplitButton.DropDownContent>
</t:RadRibbonSplitButton>
<t:RadRibbonSplitButton x:Name="SendBackwardButton" Command="{Binding
SendBackward.Command}" t:ScreenTip.Description="Send the selected object back one level so
that it's hidden behind more objects." IsEnabled="{Binding SendBackward.IsEnabled,
Mode=TwoWay}" Size="Medium" SmallImage="{t:IconResource
IconRelativePath=16/sendBackwards.png, IconSources={StaticResource IconPaths}}"
Text="Send Backward" t:ScreenTip.Title="Send Backward">
<t:RadRibbonSplitButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
```

```
<t:RadMenuItem Command="{Binding SendBackward.Command}" t:ScreenTip.Description="Send the selected object back one level so that it's hidden behind more objects." Header="Send Backward" IsEnabled="{Binding SendBackward.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Send Backward">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/sendBackwards.png, IconSources={StaticResource IconPaths}}" Stretch="None" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding SendToBack.Command}" t:ScreenTip.Description="Send the selected object behind all other objects." Header="Send to Back" IsEnabled="{Binding SendToBack.IsEnabled, Mode=TwoWay}" t:ScreenTip.Title="Send to Back">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/sendToBack.png, IconSources={StaticResource IconPaths}}" Stretch="None" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadContextMenu>
</t:RadRibbonSplitButton.DropDownContent>
</t:RadRibbonSplitButton>
<t:RadRibbonDropDownButton x:Name="RotateButton" t:ScreenTip.Description="Rotate or flip the selected object." Size="Medium" SmallImage="{t:IconResource IconRelativePath=16/rotateRight90.png, IconSources={StaticResource IconPaths}}" Text="Rotate" t:ScreenTip.Title="Rotate Objects">
<t:RadRibbonDropDownButton.DropDownContent>
<t:RadContextMenu BorderThickness="0" Padding="0">
<t:RadMenuItem Command="{Binding RotateNinetyDegreesRight.Command}" Header="Rotate Right 90°" IsEnabled="{Binding RotateNinetyDegreesRight.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/rotateRight90.png, IconSources={StaticResource IconPaths}}" Stretch="None" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding RotateNinetyDegreesLeft.Command}" Header="Rotate Left 90°" IsEnabled="{Binding RotateNinetyDegreesLeft.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/rotateLeft90.png, IconSources={StaticResource IconPaths}}" Stretch="None" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding FlipVertical.Command}" Header="Flip Vertical" IsEnabled="{Binding FlipVertical.IsEnabled, Mode=TwoWay}">
```

```
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/flipVertical.png,
IconSources={StaticResource IconPaths}}" Stretch="None" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
<t:RadMenuItem Command="{Binding FlipHorizontal.Command}" Header="Flip Horizontal"
IsEnabled="{Binding FlipHorizontal.IsEnabled, Mode=TwoWay}">
<t:RadMenuItem.Icon>
<Image Source="{t:IconResource IconRelativePath=16/flipHorizontal.png,
IconSources={StaticResource IconPaths}}" Stretch="None" Width="16"/>
</t:RadMenuItem.Icon>
</t:RadMenuItem>
</t:RadContextMenu>
</t:RadRibbonDropDownButton.DropDownContent>
</t:RadRibbonDropDownButton>
</t:RadCollapsiblePanel>
</t:RadRibbonGroup>
<t:RadRibbonGroup DialogLauncherVisibility="{Binding ShowFormatShapesDialog.IsEnabled,
Converter={StaticResource BoolToVisibilityValueConverter}}" t:ScreenTip.Description="Show the
Format Shapes dialog box in which you can specify the size and positioning fo the object."
DialogLauncherCommand="{Binding ShowFormatShapesDialog.Command}" Header="Size"
t:ScreenTip.Title="Format Shapes">
<t:RadCollapsiblePanel>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition/>
<ColumnDefinition/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Image t:ScreenTip.Description="Change the width of the shape or picture."
Source="{t:IconResource IconRelativePath=16/width.png, IconSources={StaticResource
IconPaths}}" Stretch="None" t:ScreenTip.Title="Shape Width" UseLayoutRounding="True"
VerticalAlignment="Center"/>
<TextBlock Grid.Column="1" t:ScreenTip.Description="Change the width of the shape or picture."
Margin="4 0" Text="Width:" t:ScreenTip.Title="Shape Width" VerticalAlignment="Center"/>
<t:RadNumericUpDown Grid.Column="2" t:ScreenTip.Description="Change the width of the shape
or picture." Maximum="2348" MaxWidth="150" Minimum="0.1" SmallChange="0.1"
t:ScreenTip.Title="Shape Width"
```

```
Controls:RadNumericUpDownExtensions.UIInteractionCommand="{Binding
SetShapeWidth.Command}"
Controls:RadNumericUpDownExtensions.UIInteractionCommandParameter="{Binding Value,
RelativeSource={RelativeSource Self}}" Value="{Binding SetShapeWidth.SelectedValue,
Mode=TwoWay}" VerticalAlignment="Center"/>
<Image t:ScreenTip.Description="Change the height of the shape or picture." Margin="0 2 0 0"
Grid.Row="1" Source="{t:IconResource IconRelativePath=16/height.png,
IconSources={StaticResource IconPaths}}" Stretch="None" t:ScreenTip.Title="Shape Height"
UseLayoutRounding="True" VerticalAlignment="Center"/>
<TextBlock Grid.Column="1" t:ScreenTip.Description="Change the height of the shape or picture."
Margin="4 2 4 0" Grid.Row="1" Text="Height:" t:ScreenTip.Title="Shape Height"
VerticalAlignment="Center"/>
<t:RadNumericUpDown Grid.Column="2" t:ScreenTip.Description="Change the height of the
shape or picture." Maximum="2348" MaxWidth="150" Margin="0 2 0 0" Minimum="0.1"
Grid.Row="1" SmallChange="0.1" t:ScreenTip.Title="Shape Height"
Controls:RadNumericUpDownExtensions.UIInteractionCommand="{Binding
SetShapeHeight.Command}"
Controls:RadNumericUpDownExtensions.UIInteractionCommandParameter="{Binding Value,
RelativeSource={RelativeSource Self}}" Value="{Binding SetShapeHeight.SelectedValue,
Mode=TwoWay}" VerticalAlignment="Center"/>
</Grid>
</t:RadCollapsiblePanel>
</t:RadRibbonGroup>
</t:RadRibbonTab>
</t:RadRibbonView>

<!--Formula Bar-->
<Controls:RadSpreadsheetFormulaBar Grid.Row="1" RadSpreadsheet="{Binding
ElementName=radSpreadsheet, Mode=OneTime}" Grid.ColumnSpan="2"/>

<!-- Tagged Data UI -->
<Grid Grid.Row="2" Grid.RowSpan="2" Grid.Column="1" DataContext="{Binding
ElementName=DataFormattingToolControl, Path=DataContext}">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="*"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<Grid Grid.Column="0">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
```

```
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<TextBlock Grid.Column="0" Text="Tag Type" VerticalAlignment="Center"
HorizontalAlignment="Center"/>
<TextBlock Grid.Column="1" Text="Tag Name" VerticalAlignment="Center"
HorizontalAlignment="Center"/>
<TextBlock Grid.Column="2" Text="Unit" VerticalAlignment="Center"
HorizontalAlignment="Center"/>

</Grid>

<t:RadListBox x:Name="DataTagListBox" Grid.Row="1"
EnableSelectionCaching="False"
ItemsSource="{Binding Path=Tags, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=SelectedTag, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">
<t:RadListBox.ItemTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<TextBlock Grid.Column="0" Text="{Binding Path=AttributeType }"/>
<TextBlock Grid.Column="1" Text="{Binding Path=ColumnName}"/>
<Grid Grid.Column="2" >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>

<t:RadComboBox x:Name="GeneralUnitComboBox"  Grid.Column="0" IsEnabled="{Binding
Path=IsUnitable}"
SelectedItem="{Binding Path=UnitType, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=DataContext.unitsTypes, RelativeSource={RelativeSource
Mode=FindAncestor, AncestorType={x:Type t:RadListBox}}}" />
```

```
<t:RadComboBox x:Name="PrefixedUnitComboBox" Grid.Column="1" IsEnabled="{Binding
Path=IsUnitable}"
SelectedItem="{Binding Path=Unit, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
ItemsSource="{Binding Path=Units, Mode=OneWay, UpdateSourceTrigger=PropertyChanged }">
<t:RadComboBox.Style>
<Style TargetType="t:RadComboBox">
<Setter Property="IsEnabled" Value="True"/>
<Style.Triggers>
<DataTrigger Binding="{Binding ElementName=GeneralUnitComboBox, Path=SelectedItem}"
Value="Percentage">
<Setter Property="IsEnabled" Value="False"/>
</DataTrigger>
<DataTrigger Binding="{Binding ElementName=GeneralUnitComboBox, Path=SelectedItem}"
Value="Number">
<Setter Property="IsEnabled" Value="False"/>
</DataTrigger>
</Style.Triggers>
</Style>
</t:RadComboBox.Style>
</t:RadComboBox>
</Grid>
</Grid>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>

<Grid Grid.Row="2">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="*"/>
<RowDefinition Height="*"/>
</Grid.RowDefinitions>

<t:RadButton x:Name="Build_DigitalModes_Btn" Grid.Row="0" Grid.Column="0" Padding="10"
Content="Build RFFE Pattern" Command="{Binding Path=BuildDigitalPatternModelCommand}"
CommandParameter="{Binding RelativeSource={RelativeSource
AncestorType=t:RadWindow,Mode=FindAncestor}}"/>
<t:RadButton x:Name="Build_Tables_Btn" Grid.Row="0" Grid.Column="1" Padding="10"
Content="Build Tests" Command="{Binding Path=ExportTablesCommand}"
```

```
CommandParameter="{Binding RelativeSource={RelativeSource
AncestorType=t:RadWindow,Mode=FindAncestor}}"/>
<t:RadButton x:Name="Delete_Tag_Btn" Grid.Row="1" Grid.Column="0" Content="Delete Tag"
Padding="10" Command="{Binding Path=DeleteTagCommand}"/>
<t:RadButton x:Name="Reset_Tags_Btn" Grid.Row="1" Grid.Column="1" Content="Reset Tags"
Padding="10" Command="{Binding Path=ResetTagsCommand}"/>

</Grid>
</Grid>

<!--Spreadsheet Workspace-->
<t:RadSpreadsheet x:Name="radSpreadsheet" Grid.Row="2" Grid.Column="0"
DataContext="{Binding ElementName=DataFormattingToolControl, Path=DataContext}"
Workbook="{Binding Path=DataWorkBook, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}">
<t:EventToCommandBehavior.EventBindings>
<t:EventBinding EventName="MouseLeftButtonUp" Command="{Binding
Path=SelectionChangedCommand}" CommandParameter="{Binding
ElementName=radSpreadsheet, Path=ActiveSheetEditor.Selection}"
RaiseOnHandledEvents="True"/>
</t:EventToCommandBehavior.EventBindings>
<t:RadSpreadsheet.FormatProviders>
<Txt:TxtFormatProvider/>
<Csv:CsvFormatProvider/>
<Pdf:PdfFormatProvider/>
<Xlsx:XlsxFormatProvider/>
</t:RadSpreadsheet.FormatProviders>
</t:RadSpreadsheet>

<!--Status Bar-->
<Controls:RadSpreadsheetStatusBar Grid.Row="3" RadSpreadsheet="{Binding
ElementName=radSpreadsheet, Mode=OneTime}" Grid.Column="0"/>
</Grid>
</UserControl>
```

FunctionSequencingTool.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Tools.FunctionSequencingTool"
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:MerlinTestStudio_Demo_Telerik.UserControls.Tools"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
xmlns:dd="clr-namespace:MerlinTestStudio_Demo_Telerik.Data"
xmlns:TemplateSelector1="clr-
namespace:MerlinTestStudio_Demo_Telerik.Data.Selectors.DataTemplateSelectors"
xmlns:TemplateSelector="clr-
namespace:MerlinTestStudio_Demo_Telerik.Data.DataTemplateSelectors"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<t:BooleanToVisibilityConverter x:Key="BooltoVisConverter"/>
<TemplateSelector:DUTSequenceItemSelector x:Key="ActionSequenceItemSelecter" />
<TemplateSelector1:ParameterTemplateSelector x:Key="ParameterValueEditorTemplateSelector"
/>

<!--ListBox Headers-->
<Style x:Key="FuctionSelectionListBoxHeader" TargetType="TextBlock">
<Setter Property="HorizontalAlignment" Value="Center"/>
<Setter Property="VerticalAlignment" Value="Center"/>
<Setter Property="FontSize" Value="20"/>
<Setter Property="Width" Value="Auto"/>
<Setter Property="Height" Value="Auto"/>
<Setter Property="Foreground" Value="LightGray"/>
</Style>

<!--Flipped Image Style-->
<Style TargetType="Image" x:Key="FlippedImage">
<Setter Property="RenderTransformOrigin" Value="0.5,0.5"/>
<Setter Property="RenderTransform">
<Setter.Value>
<TransformGroup>
<RotateTransform Angle="180"/>
</TransformGroup>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="MenuItemContainerStyle" TargetType="t:RadMenuItem">
<Setter Property="Header" Value="{Binding Path=Text}"          />
<Setter Property="ItemsSource" Value="{Binding Path=SubItems}"/>
<Setter Property="IsEnabled" Value="{Binding Path=IsEnabled}"/>
```

```
</Style>

<Style x:Key="BetterStyle" TargetType="t:RadListBoxItem">
<Setter Property="t:DragDropManager.AllowCapturedDrag" Value="True"/>
</Style>
</UserControl.Resources>

<Grid>

<!--Sequence Item Display-->
<Grid >
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="30"/>
<RowDefinition Height="*"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>

<Grid Grid.Row="0" Grid.Column="0">
<TextBlock Text="Function Sequence" Style="{StaticResource FuctionSelectionListBoxHeader}"/>
</Grid>
<t:RadListBox x:Name="FunctionSequenceListDUT" Grid.Row="1" Grid.Column="0"
ItemContainerStyle="{StaticResource BetterStyle}"
ItemTemplateSelector="{StaticResource ActionSequenceItemSelecter}"
ItemsSource="{Binding PassedSequenceCollection, RelativeSource={RelativeSource
AncestorType=UserControl}}"
EnableSelectionCaching="False"
AllowDrop="True">

<t:RadListBox.DragDropBehavior >
<dd:CustomDragDropBehaviorForFSL/>
</t:RadListBox.DragDropBehavior>
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>

</t:RadListBox>

<Grid Grid.Row="0" Grid.Column="1">
```

```
<TextBlock Text="Function Library" Style="{StaticResource FuctionSelectionListBoxHeader}"/>
</Grid>
<Grid Grid.Row="1" Grid.Column="1">
<Grid.RowDefinitions>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="3*"/>
</Grid.RowDefinitions>

<t:RadMenu Grid.Row="0">
<t:RadMenuItem x:Name="ImportBtn" Header="Import DLL" Click="ImportBtn_Click"/>
<t:RadMenuSeparatorItem/>
</t:RadMenu>

<StackPanel Grid.Row="1" Margin="5">
<TextBlock Text="Source: "/>
<t:RadComboBox x:Name="SourceComboBox" >
<t:RadComboBox.ItemTemplate>
<DataTemplate x:Name="cmbDTem">
<StackPanel Orientation="Horizontal">
<TextBlock Name="Item" Text="{Binding Path=NameOfDLL}" Width="Auto"/>
</StackPanel>
</DataTemplate>
</t:RadComboBox.ItemTemplate>
</t:RadComboBox>
</StackPanel>

<t:RadListBox x:Name="FunctionsLibraryList"
Grid.Row="2"
ItemContainerStyle="{StaticResource DraggableListBoxItem}"
AllowDrop="False"
EnableSelectionCaching="False">
<t:RadListBox.Style>
<Style TargetType="t:RadListBox">
<Setter Property="ItemsSource" Value="{Binding ElementName=SourceComboBox,
Path=SelectedValue.DLLFunctions}"/>
</Style>
</t:RadListBox.Style>

<t:RadListBox.DragDropBehavior >
<dd:CustomDragDropBehaviorForFFL />
</t:RadListBox.DragDropBehavior>
```

```
<t:RadListBox.DragVisualProvider>
<t:ScreenshotDragVisualProvider />
</t:RadListBox.DragVisualProvider>

</t:RadListBox>

</Grid>

<WrapPanel Grid.Row="2" Grid.ColumnSpan="2" HorizontalAlignment="Center" Height="Auto"
Margin="10">
<t:RadButton x:Name="CancelBtn" Content="Cancel" Margin="0 0 40 0" Width="140" Height="30"
Click="CancelBtn_Click" />
<t:RadButton x:Name="DoneBtn" Content="Done" Width="140" Height="30"
Command="t:WindowCommands.Close"/>
</WrapPanel>

</Grid>

</Grid>
</UserControl>
```

GeneratePinsFromCalibrationDefinition.xaml

```
<UserControl
x:Class="MerlinTestStudio_Demo_Telerik.UserControls.Tools.GeneratePinsFromCalibrationDefini
tion"
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:MerlinTestStudio_Demo_Telerik.UserControls.Tools"
xmlns:t="http://schemas.telerik.com/2008/xaml/presentation"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800">

<UserControl.Resources>
<Style x:Key="SelectionListBoxHeader" TargetType="TextBlock">
<Setter Property="HorizontalAlignment" Value="Center"/>
<Setter Property="VerticalAlignment" Value="Center"/>
<Setter Property="FontSize" Value="20"/>
<Setter Property="Width" Value="Auto"/>
<Setter Property="Height" Value="Auto"/>
<Setter Property="Foreground" Value="LightGray"/>
```

```
</Style>
</UserControl.Resources>

<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition Height="40"/>
<RowDefinition Height="*"/>
<RowDefinition Height="40"/>
</Grid.RowDefinitions>

<Grid Grid.Row="0" Grid.ColumnSpan="2">
<TextBlock Text="Parameters" Style="{StaticResource SelectionListBoxHeader}"/>
</Grid>

<t:RadListBox x:Name="AutoGenList" Grid.Row="1" Grid.ColumnSpan="2" Margin="10"
ItemsSource="{Binding Path=AutoGeneratedParameters, Mode=OneWay,
UpdateSourceTrigger=PropertyChanged}">

<t:RadListBox.ItemTemplate>
<DataTemplate>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="2*"/>
</Grid.ColumnDefinitions>

<CheckBox Grid.Column="0" Content="Import" IsChecked="{Binding Path=IsAutoGen,
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
<TextBlock Grid.Column="1" Text="{Binding Path=ColumnName}"/>

</Grid>
</DataTemplate>
</t:RadListBox.ItemTemplate>
</t:RadListBox>

<!--Buttons Area-->
<WrapPanel Grid.Row="2" Grid.ColumnSpan="2" HorizontalAlignment="Center">
<t:RadButton Content="Cancel" Margin="0 0 40 0" Width="140" Height="30"
```

```
Command="{Binding Path=CancelBtnCommand}"
CommandParameter="{Binding RelativeSource={RelativeSource
AncestorType=t:RadWindow,Mode=FindAncestor}}">
</t:RadButton>
<t:RadButton Content="Done" Width="140" Height="30"
Command="{Binding Path=DoneBtnCommand}"
CommandParameter="{Binding RelativeSource={RelativeSource
AncestorType=t:RadWindow,Mode=FindAncestor}}">
</t:RadButton>
</WrapPanel>
</Grid>
</UserControl>
```

WaveformConverterControl.xaml

```
<UserControl x:Class="WaveformConverterControls.WaveformConverterControl"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:WaveformConverterControls"
xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
xmlns:Charts="clr-namespace:WaveformConverterControls"
mc:Ignorable="d"
TextElement.Foreground="{DynamicResource MaterialDesignBody}"
TextElement.FontWeight="Regular"
TextElement.FontSize="14"
TextOptions.TextFormattingMode="Ideal"
TextOptions.TextRenderingMode="Auto"
Background="{DynamicResource MaterialDesignPaper}"
FontFamily="{DynamicResource MaterialDesignFont}"
d:DesignHeight="1080" d:DesignWidth="1920" Loaded="UserControl_Loaded">

<UserControl.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Dark.xaml" />
<ResourceDictionary
Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Defaults.xaml" />
<ResourceDictionary
```

```
Source="pack://application:,,,/MaterialDesignColors;component/Themes/Recommended/Primary/
MaterialDesignColor.grey.xaml" />
<ResourceDictionary
Source="pack://application:,,,/MaterialDesignColors;component/Themes/Recommended/Accent/M
aterialDesignColor.Lime.xaml" />
</ResourceDictionary.MergedDictionaries>
<!--This style prevents dashed bottoms of disabled textboxes from hanging around when you don't
want them to in the Metadata tab-->
<Style TargetType="materialDesign:BottomDashedLineAdorner">
<Setter Property="Visibility" Value="Collapsed"/>
</Style>

<Style x:Key="DialogButtonStyle" TargetType="Button">
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
<Grid>
<Rectangle x:Name="GelBackground" Opacity="1" RadiusX="9" RadiusY="9"
Fill="{TemplateBinding Background}" StrokeThickness="0.35">
<Rectangle.Stroke>
<LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
<GradientStop Color="#FF6495ED" Offset="0" />
<GradientStop Color="#FF6495ED" Offset="1" />
</LinearGradientBrush>
</Rectangle.Stroke>
</Rectangle>
<Rectangle x:Name="GelShine" Margin="2,2,2,0" VerticalAlignment="Top" RadiusX="6"
RadiusY="6"
Opacity="1" Stroke="Transparent" Height="15px">
<Rectangle.Fill>
<LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
<GradientStop Color="#FF6495ED" Offset="0"/>
<GradientStop Color="Transparent" Offset="1"/>
</LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
<ContentPresenter VerticalAlignment="Center" HorizontalAlignment="Center"/>
</Grid>
<ControlTemplate.Triggers>
<Trigger Property="IsMouseOver" Value="True">
<Setter Property="Background" Value="Brown">
```

```
</Setter>
</Trigger>
<Trigger Property="IsPressed" Value="True">
<Setter Property="Fill" TargetName="GelBackground">
<Setter.Value>
<LinearGradientBrush EndPoint="0,1" StartPoint="0,0">
<GradientStop Color="Blue" Offset="0"/>
<GradientStop Color="Blue" Offset="1"/>
</LinearGradientBrush>
</Setter.Value>
</Setter>
</Trigger>
<Trigger Property="IsEnabled" Value="False">
<Setter Property="Fill" TargetName="GelBackground" Value="LightGray">

</Setter>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="Background" Value="#FF4169E1"/>
<Setter Property="Foreground" Value="White"/>
<Setter Property="Width" Value="55"/>
<Setter Property="Height" Value="30"/>
</Style>

<Style x:Key="ColumnElementStyle" TargetType="TextBlock">
<Setter Property="Margin" Value="0,-5,0,-5" />
</Style>

<ControlTemplate x:Key="rowValidationErrorTemplate">
<Grid Margin="0,-2,0,-2" ToolTip="{Binding RelativeSource={RelativeSource FindAncestor,
AncestorType={x:Type DataGridRow}},Path=(Validation.Errors)[0].ErrorContent}">
<Ellipse StrokeThickness="0" Fill="Red" Width="{TemplateBinding FontSize}"
Height="{TemplateBinding FontSize}" />
<TextBlock Text="!" FontSize="{TemplateBinding FontSize}" FontWeight="Bold"
Foreground="White" HorizontalAlignment="Center" VerticalAlignment="Center" />
</Grid>
</ControlTemplate>

<local:BindingProxy x:Key="proxy" Data="{Binding}"/>
```

```
</ResourceDictionary>
</UserControl.Resources>

<DockPanel LastChildFill="True">
<!--Create a grid to hold the Loop and Index Time.-->
<Grid Margin="5">

<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
<RowDefinition Height="200" />
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="Auto" />
</Grid.ColumnDefinitions>

<Border BorderBrush="{DynamicResource PrimaryHueLightBrush}" BorderThickness="1"
Margin="5,5,5,10" DockPanel.Dock="Top" Grid.Column="0" Grid.Row="1" Grid.RowSpan="2">
<DockPanel Margin="5" >
<Grid DockPanel.Dock="Top" ShowGridLines="False">
<!--Create a grid to hold the label and textbox-->
<Grid.RowDefinitions>
<RowDefinition Height="*" />
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>
<ColumnDefinition Width="*" />
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="Auto" MinWidth="350"/>
</Grid.ColumnDefinitions>

<Charts:XvsYGenericChart x:Name="panel1Chart" ChartType="FastLine" Grid.Row="0"
Grid.Column="0" Margin="0,0,0,10" />

<GridSplitter Width="5" Grid.Row="0" Grid.Column="1" HorizontalAlignment="Stretch"
Background="Transparent" />

<DockPanel Grid.Row="0" Grid.Column="2">
```

```
<TabControl Background="{DynamicResource MaterialDesignPaper}" x:Name="myTab"
Grid.Row="0" Grid.Column="0" SelectedIndex="{Binding
TabSelectedIndex,Mode=OneWayToSource}" SelectedItem="{Binding
TabSelected,Mode=OneWayToSource}">
<TabItem Header="Marker1">
<DockPanel>
<StackPanel Orientation="Horizontal" DockPanel.Dock="Bottom">
<Button Style="{DynamicResource DialogButtonStyle}" Content="Restore" Command="{Binding
RestoreCommand}" Margin="270 0 0 0" />
<Button Style="{DynamicResource DialogButtonStyle}" Content="Apply" Command="{Binding
ApplyCommand}" Margin="10 0 0 0" />
</StackPanel>
<DataGrid DockPanel.Dock="Top" Name="dgMarker1" AutoGenerateColumns="False"
ItemsSource="{Binding Marker1Segments}" SelectionMode="Single"
RowValidationErrorTemplate="{StaticResource rowValidationErrorTemplate}"
HeadersVisibility="All" Margin="0,3,0,0" ScrollViewer.VerticalScrollBarVisibility="Auto">
<DataGrid.RowValidationRules>
<local:MarkerSegmentValidationRule ValidationStep="UpdatedValue">
<local:MarkerSegmentValidationRule.ValidationData>
<local:MarkerSegmentValidationData LowerLimit="1" UpperLimit="{Binding
Path=Data.NumberOfSamples,Source={StaticResource proxy}}" />
</local:MarkerSegmentValidationRule.ValidationData>
</local:MarkerSegmentValidationRule>
</DataGrid.RowValidationRules>
<DataGrid.Columns>
<DataGridTextColumn Header="From" Width="150" Binding="{Binding From}" />
<DataGridTextColumn Header="To" Width="150" Binding="{Binding To}" />
</DataGrid.Columns>
</DataGrid>
</DockPanel>
</TabItem>
<TabItem Header="Marker2">
<DockPanel>
<StackPanel Orientation="Horizontal" DockPanel.Dock="Bottom">
<Button Style="{DynamicResource DialogButtonStyle}" Content="Restore" Command="{Binding
RestoreCommand}" Margin="270 0 0 0" />
<Button Style="{DynamicResource DialogButtonStyle}" Content="Apply" Command="{Binding
ApplyCommand}" Margin="10 0 0 0" />
</StackPanel>
<DataGrid DockPanel.Dock="Top" Name ="dgMarker2" AutoGenerateColumns="False"
ItemsSource="{Binding Marker2Segments}" SelectionMode="Single"
RowValidationErrorTemplate="{StaticResource rowValidationErrorTemplate}"
```

```
HeadersVisibility="All" ScrollViewer.VerticalScrollBarVisibility="Auto">
<DataGrid.RowValidationRules>
<local:MarkerSegmentValidationRule ValidationStep="UpdatedValue">
<local:MarkerSegmentValidationRule.ValidationData>
<local:MarkerSegmentValidationData LowerLimit="1" UpperLimit="{Binding
Path=Data.NumberOfSamples,Source={StaticResource proxy}}" />
</local:MarkerSegmentValidationRule.ValidationData>
</local:MarkerSegmentValidationRule>
</DataGrid.RowValidationRules>
<DataGrid.Columns>
<DataGridTextColumn Header="From" Width="150" Binding="{Binding From}" />
<DataGridTextColumn Header="To" Width="150" Binding="{Binding To}" />
</DataGrid.Columns>
</DataGrid>
</DockPanel>
</TabItem>
<TabItem Header="Marker3">
<DockPanel LastChildFill="True">
<StackPanel Orientation="Horizontal" DockPanel.Dock="Bottom">
<Button Style="{DynamicResource DialogButtonStyle}" Content="Restore" Command="{Binding
RestoreCommand}" Margin="270 0 0 0" />
<Button Style="{DynamicResource DialogButtonStyle}" Content="Apply" Command="{Binding
ApplyCommand}" Margin="10 0 0 0" />
</StackPanel>
<DataGrid DockPanel.Dock="Top" Name ="dgMarker3" AutoGenerateColumns="False"
ItemsSource="{Binding Marker3Segments}" SelectionMode="Single"
RowValidationErrorTemplate="{StaticResource rowValidationErrorTemplate}"
HeadersVisibility="All" ScrollViewer.VerticalScrollBarVisibility="Auto">
<DataGrid.RowValidationRules>
<local:MarkerSegmentValidationRule ValidationStep="UpdatedValue">
<local:MarkerSegmentValidationRule.ValidationData>
<local:MarkerSegmentValidationData LowerLimit="1" UpperLimit="{Binding
Path=Data.NumberOfSamples,Source={StaticResource proxy}}" />
</local:MarkerSegmentValidationRule.ValidationData>
</local:MarkerSegmentValidationRule>
</DataGrid.RowValidationRules>
<DataGrid.Columns>
<DataGridTextColumn Header="From" Width="150" Binding="{Binding From}" />
<DataGridTextColumn Header="To" Width="150" Binding="{Binding To}" />
</DataGrid.Columns>
</DataGrid>
</DockPanel>
```

```
</TabItem>
<TabItem Header="Marker4">
<DockPanel LastChildFill="True">
<StackPanel Orientation="Horizontal" DockPanel.Dock="Bottom">
<Button Style="{DynamicResource DialogButtonStyle}" Content="Restore" Command="{Binding
RestoreCommand}" Margin="270 0 0 0" />
<Button Style="{DynamicResource DialogButtonStyle}" Content="Apply" Command="{Binding
ApplyCommand}" Margin="10 0 0 0" />
</StackPanel>
<DataGrid DockPanel.Dock="Top" Name ="dgMarker4" AutoGenerateColumns="False"
ItemsSource="{Binding Marker4Segments}" SelectionMode="Single"
RowValidationErrorTemplate="{StaticResource rowValidationErrorTemplate}"
HeadersVisibility="All" ScrollViewer.VerticalScrollBarVisibility="Auto">
<DataGrid.RowValidationRules>
<local:MarkerSegmentValidationRule ValidationStep="UpdatedValue">
<local:MarkerSegmentValidationRule.ValidationData>
<local:MarkerSegmentValidationData LowerLimit="1" UpperLimit="{Binding
Path=Data.NumberOfSamples,Source={StaticResource proxy}}" />
</local:MarkerSegmentValidationRule.ValidationData>
</local:MarkerSegmentValidationRule>
</DataGrid.RowValidationRules>
<DataGrid.Columns>
<DataGridTextColumn Header="From" Width="150" Binding="{Binding From}" />
<DataGridTextColumn Header="To" Width="150" Binding="{Binding To}" />
</DataGrid.Columns>
</DataGrid>
</DockPanel>
</TabItem>
<TabItem Header="Metadata">
<DockPanel LastChildFill="True">
<StackPanel Orientation="Horizontal" DockPanel.Dock="Bottom">
<!--Hiding metadata checkbox for now but leaving code in case we'd like to expose it later.-->
<CheckBox IsChecked="{Binding MetadataEditingEnabled}" Foreground="WhiteSmoke"
Content="Enable Metadata Editing" Visibility="Hidden"></CheckBox>
</StackPanel>
<ScrollViewer VerticalScrollBarVisibility="Auto">
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
```

```
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Label Grid.Row="0" Grid.Column="0" Content="Format" Visibility ="Visible" IsEnabled="{Binding
MetadataEditingEnabled}"/>
<Label Grid.Row="1" Grid.Column="0" Content="Signal Bandwidth (MHz)" Visibility="Visible"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="2" Grid.Column="0" Content="Sampling Rate (MHz)" Visibility="Visible"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="3" Grid.Column="0" Content="Number of Samples" Visibility="Visible"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="4" Grid.Column="0" Content="Duty Cycle (%)" Visibility="Visible"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="5" Grid.Column="0" Content="Crest Factor (dB)" Visibility="Visible"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="6" Grid.Column="0" Content="Rel RMS" Visibility="Visible"
```

```
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="7" Grid.Column="0" Content="Total Time (ms)" Visibility="Visible"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="8" Grid.Column="0" Content="Offset (ms)" Visibility="Visible"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="9" Grid.Column="0" Content="Measurement Length (ms)" Visibility="Visible"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="10" Grid.Column="0" Content="Step Length (ms)" Visibility="Visible"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="11" Grid.Column="0" Content="Number of Steps" Visibility="Visible"
IsEnabled="{Binding MetadataEditingEnabled}"/>

<!--NR specific-->
<Label Grid.Row="12" Grid.Column="0" Content="Subcarrier Spacing (kHz)" Visibility="{Binding
NRVis}" IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="13" Grid.Column="0" Content="Cyclic Prefix" Visibility="{Binding NRVis}"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="14" Grid.Column="0" Content="Transform Precoding" Visibility="{Binding
NRVis}" IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="15" Grid.Column="0" Content="PUSCH Modulation" Visibility="{Binding
NRVis}" IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="16" Grid.Column="0" Content="PUSCH Slots" Visibility="{Binding NRVis}"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="17" Grid.Column="0" Content="PUSCH Resource Blocks" Visibility="{Binding
NRVis}" IsEnabled="{Binding MetadataEditingEnabled}"/>
<!--WLAN specific-->
<Label Grid.Row="18" Grid.Column="0" Content="Guard Interval (us)" Visibility="{Binding
WLANVis}" IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="19" Grid.Column="0" Content="MCS" Visibility="{Binding WLANVis}"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="20" Grid.Column="0" Content="Data Rate" Visibility="{Binding WLANVis}"
IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="21" Grid.Column="0" Content="LTF Compression Type" Visibility="{Binding
WLANVis}" IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="22" Grid.Column="0" Content="Data Symbols" Visibility="{Binding WLANVis}"
IsEnabled="{Binding MetadataEditingEnabled}"/>

<!--C2K specific-->
<Label Grid.Row="23" Grid.Column="0" Content="Radio Config" Visibility="{Binding C2KVis}"
IsEnabled="{Binding MetadataEditingEnabled}"/>

<!--1xEVDO specific-->
```

```
<Label Grid.Row="24" Grid.Column="0" Content="Physical Layer Subtype" Visibility="{Binding EVDOVis}" IsEnabled="{Binding MetadataEditingEnabled}"/>

<!--LTE DL specific-->
<Label Grid.Row="25" Grid.Column="0" Content="Downlink Mode" Visibility="{Binding LTEDownlinkVis}" IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="26" Grid.Column="0" Content="UL/DL Config Index" Visibility="{Binding LTEDownlinkVis}" IsEnabled="{Binding MetadataEditingEnabled}"/>
<Label Grid.Row="27" Grid.Column="0" Content="PDCCH Num Symbol" Visibility="{Binding LTEDownlinkVis}" IsEnabled="{Binding MetadataEditingEnabled}"/>



<ComboBox Foreground="WhiteSmoke" Grid.Row="0" Grid.Column="1" SelectedItem="{Binding ComboboxRadioFormatSelectedValue}" ItemsSource="{Binding ComboboxRadioFormatList}" SelectedValuePath ="Key" DisplayMemberPath="Value" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"></ComboBox>
<TextBox Foreground="WhiteSmoke" Grid.Row="2" Grid.Column="1" Text="{Binding SamplingRate}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="3" Grid.Column="1" Text="{Binding NumberOfSamples}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="4" Grid.Column="1" Text="{Binding DutyCycle, StringFormat=0.###}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="5" Grid.Column="1" Text="{Binding CrestFactor, StringFormat=0.###}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="1" Grid.Column="1" Text="{Binding SignalBandwidth}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="6" Grid.Column="1" Text="{Binding RelRms, StringFormat=0.###}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="7" Grid.Column="1" Text="{Binding TotalTime, StringFormat=0.###}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="8" Grid.Column="1" Text="{Binding Offset, StringFormat=0.###}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="9" Grid.Column="1" Text="{Binding MeasLength, StringFormat=0.###}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="10" Grid.Column="1" Text="{Binding StepLength, StringFormat=0.###}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="11" Grid.Column="1" Text="{Binding NumSteps}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="Visible"/>
```

<ComboBox Foreground="WhiteSmoke" Grid.Row="12" Grid.Column="1" SelectedItem="{Binding ComboboxSubcarrierSpacingSelectedValue}" ItemsSource="{Binding ComboboxSubcarrierSpacingList}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding NRVis}"></ComboBox>
<ComboBox Foreground="WhiteSmoke" Grid.Row="13" Grid.Column="1" SelectedItem="{Binding ComboboxCyclicPrefixSelectedValue}" ItemsSource="{Binding ComboboxCyclicPrefixList}" SelectedValuePath ="Key" DisplayMemberPath="Value" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding NRVis}"></ComboBox>
<ComboBox Foreground="WhiteSmoke" Grid.Row="14" Grid.Column="1" SelectedItem="{Binding ComboboxTransformPrecodingSelectedValue}" ItemsSource="{Binding ComboboxTransformPrecodingList}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding NRVis}"></ComboBox>
<ComboBox Foreground="WhiteSmoke" Grid.Row="15" Grid.Column="1" SelectedItem="{Binding ComboboxPUSCHModSelectedValue}" ItemsSource="{Binding ComboboxPUSCHModList}" SelectedValuePath ="Key" DisplayMemberPath="Value" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding NRVis}"></ComboBox>
<TextBox Foreground="WhiteSmoke" Grid.Row="16" Grid.Column="1" Text="{Binding PUSCHSlots}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding NRVis}"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="17" Grid.Column="1" Text="{Binding PUSCHPRB}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding NRVis}"/>

<ComboBox Foreground="WhiteSmoke" Grid.Row="18" Grid.Column="1" SelectedItem="{Binding ComboboxGuardIntervalSelectedValue}" ItemsSource="{Binding ComboboxGuardIntervalList}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding WLANVis}"></ComboBox>
<ComboBox Foreground="WhiteSmoke" Grid.Row="19" Grid.Column="1" SelectedItem="{Binding ComboboxMCSSelectedValue}" ItemsSource="{Binding ComboboxMCSList}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding WLANVis}"></ComboBox>
<ComboBox Foreground="WhiteSmoke" Grid.Row="20" Grid.Column="1" SelectedItem="{Binding ComboboxDataRateSelectedValue}" ItemsSource="{Binding ComboboxDataRateList}" SelectedValuePath ="Key" DisplayMemberPath="Value" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding WLANVis}"></ComboBox>
<ComboBox Foreground="WhiteSmoke" Grid.Row="21" Grid.Column="1" SelectedItem="{Binding ComboboxLTFCompTypeSelectedValue}" ItemsSource="{Binding ComboboxLTFCompTypeList}" SelectedValuePath ="Key" DisplayMemberPath="Value" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding WLANVis}"></ComboBox>
<TextBox Foreground="WhiteSmoke" Grid.Row="22" Grid.Column="1" Text="{Binding DataSymbols}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding WLANVis}"/>

<ComboBox Foreground="WhiteSmoke" Grid.Row="23" Grid.Column="1" SelectedItem="{Binding ComboboxRadioConfigSelectedValue}" ItemsSource="{Binding ComboboxRadioConfigList}" SelectedValuePath ="Key" DisplayMemberPath="Value" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding C2KVis}"></ComboBox>

```
<ComboBox Foreground="WhiteSmoke" Grid.Row="24" Grid.Column="1" SelectedItem="{Binding ComboboxPhysLayerSubtypeSelectedValue}" ItemsSource="{Binding ComboboxPhysLayerSubtypeList}" SelectedValuePath ="Key" DisplayMemberPath="Value" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding EVDOVis}"></ComboBox>

<ComboBox Foreground="WhiteSmoke" Grid.Row="25" Grid.Column="1" SelectedItem="{Binding ComboboxLTEDownlinkModeSelectedValue}" ItemsSource="{Binding ComboboxLTEDownlinkModeList}" SelectedValuePath ="Key" DisplayMemberPath="Value" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding LTEDownlinkVis}"></ComboBox>
<TextBox Foreground="WhiteSmoke" Grid.Row="26" Grid.Column="1" Text="{Binding ULDLConfigIndex}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding LTEDownlinkVis}"/>
<TextBox Foreground="WhiteSmoke" Grid.Row="27" Grid.Column="1" Text="{Binding PDCCHNumSymbol}" IsEnabled="{Binding MetadataEditingEnabled}" Visibility="{Binding LTEDownlinkVis}"/>
</Grid>
</ScrollViewer>
</DockPanel>
</TabItem>
</TabControl>
</DockPanel>
</Grid>
</DockPanel>
</Border>

</Grid>
</DockPanel>
</UserControl>
```

XvsYGenericChart.xaml

```
<UserControl x:Class="WaveformConverterControls.XvsYGenericChart"
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:winformchart="clr-namespace:System.Windows.Forms.DataVisualization.Charting;assembly=System.Windows.Forms.DataVisualization"
xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
mc:Ignorable="d"
```

```
d:DesignHeight="600" d:DesignWidth="800">
<UserControl.Resources>
<ResourceDictionary>
<ResourceDictionary.MergedDictionaries>
</ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
</UserControl.Resources>
<DockPanel LastChildFill="True">
<!--WPF control to hold a windows form control. This is used to hold the MSChart control-->
<WindowsFormsHost x:Name="host" Margin="12,6,12,6" DockPanel.Dock="Bottom" >
<winformchart:Chart x:Name="XvsYChart" Dock="Fill" MouseDown="XvsYChart_MouseDown"
AntiAliasing="All" TextAntiAliasingQuality="High" BackColor="#303030">
</winformchart:Chart>
</WindowsFormsHost>
</DockPanel>
</UserControl>
```

App.xaml.cs

```
using System.Collections.ObjectModel;
using System.Windows;
using System.Windows.Data;
using Telerik.Windows.Controls;
using Unity;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Unity.Lifetime;
using System.ComponentModel;
using MerlinTestStudio_Demo_Telerik.UserControls.TestConditions;
using MerlinTestStudio_Demo_Telerik.Data.Services;
using MT.TestStudio.GUI.ViewModels;
using UnitConversionLib;
using Telerik.Windows.Input.Touch;
using Telerik.Windows.Automation.Peers;
using System.IO;
using MerlinTestStudio_Demo_Telerik.Data;
using System.Windows.Controls;
using System;
using System.Windows.Input;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;

namespace MerlinTestStudio_Demo_Telerik
{
```

```
/// <summary>
/// Interaction logic for App.xaml
/// </summary>
public partial class App : Application
{
public UnityContainer Container { get; set; }
public MainWindowViewModel MWVM { get; set; }

protected override void OnStartup(StartupEventArgs e)
{
//base.OnStartup(e);
#region Unity IoC dependency injection
Container = new UnityContainer();

//Register ViewModels
Container.RegisterType<MainWindowViewModel>();
Container.RegisterType<ConditionsViewModel>();
Container.RegisterType<PinMapViewModel>();
Container.RegisterType<DUT_TestViewModel>();
Container.RegisterType<SystemConfigViewModel>();
Container.RegisterType<ConnectionDiagramViewModel>();
Container.RegisterType<DataFormattingToolViewModel>();

//Register Services
Container.RegisterType<IViewModelLocator, ViewModelLocator>();
Container.RegisterType<IServiceLocator, ServiceLocator>();
Container.RegisterType<ITestParametersService, TestParametersService>();
Container.RegisterType<ISequenceService, SequenceService>();
Container.RegisterType<IExtensionService, ExtensionService>();
Container.RegisterType<IPinMapService, PinMapService>();
Container.RegisterType<IInstrumentService, InstrumentService>();
Container.RegisterType<IProjectConfigurationService, ProjectConfigurationService>();
Container.RegisterType<IDataFormattingService, DataFormattingService>();
Container.RegisterType<IParameterValueService, ParameterValueService>();
Container.RegisterType<IUnitService, UnitService>();
Container.RegisterType<ITestService, TestService>();
//Container.RegisterType<INotifyPropertyChanged>();

Current.Resources.Add("IoC", Container);
#endregion Unity IoC dependency injection

//Creates all unit conversion definitions to be used everywhere in the application
```

```
UnitHelper.CreateUnits();

#region Set Application Settings and preferences
TouchManager.IsTouchEnabled = false; //For GridView speed purposes.
AutomationManager.AutomationMode = AutomationMode.Disabled; //For GridView speed
purposes.

//InitializeComponent();

//Will need to be stored in preferences later.
StyleManager.ApplicationTheme = new VisualStudio2013Theme();
VisualStudio2013Palette.LoadPreset(VisualStudio2013Palette.ColorVariation.Dark);

//VisualStudio2013Palette.Palette.DefaultForegroundColor =
System.Windows.Media.Color.FromRgb(255, 255, 255); //Work around.
#endregion Set Application Settings and preferences

#region Required Directory Checks
//Deserialize system (systems collection in the future).
if (!Directory.Exists(@"C:\MerlinTest\System Configs\"))
{
Directory.CreateDirectory(@"C:\MerlinTest\System Configs\");
}
//Check the MTS AppData Folder exists before trying to save or load files from it.
if
(!Directory.Exists(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Applicati
onData), "Merlin Test Studio")))
{
Directory.CreateDirectory(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.
ApplicationData), "Merlin Test Studio"));
}
# endregion Required Directory Checks

///Not needed any more...
DataManagement.InitializeDataManger();

#region Window creation and Project load
MainWindow wnd = new MainWindow(); //Create new window manually.
MWVM = (wnd.DataContext as MainWindowViewModel); //Take the data context and store it.
DataStorage.MainViewModel = MWVM;
//Current.Resources.Add("MainVM", MWVM); //set the application resource using the stored data
context.
```

```
if (e.Args.Length > 0) //Check to see if user opened app from project file via args.
{
switch (Path.GetExtension(e.Args[0]))
{
case BackendConstants.MerlinProjectExtension:
MWVM.LoadMerlinProject(e.Args[0]);
break;
case ".mtp":
MWVM.LoadLegacyProjectMTP(e.Args[0]);  //Convert project.
break;
default: //Do Nothing
break;
}
}
wnd.Show();
#endregion Window creation and Project load

//KeyBinding keyBinding = new KeyBinding(RadDockingCommands.ClosePane, new
KeyGesture(Key.F4, ModifierKeys.Control)) { CommandParameter =
ClosePaneMode.DocumentPanes };
//CommandManager.RegisterClassInputBinding(typeof(RadDocking), keyBinding);
}
}
}
```

MainWindow.xaml.cs

```
using System;
using System.Windows;
using Telerik.Windows.Controls;
using MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager;
using MerlinTestStudio_Demo_Telerik.UserControls.TestConditions;
using MerlinTestStudio_Demo_Telerik.UserControls.TestLimits;
using MerlinTestStudio_Demo_Telerik.UserControls.TestSequences;
using MerlinTestStudio_Demo_Telerik.UserControls;
using System.Windows.Input;
using System.Windows.Controls;
using MT.TestStudio.GUI;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Windows.Media;
using System.Windows.Data;
using System.Windows.Media.Imaging;
using MT.TestStudio.GUI.User_Controls;
```

```
using MT.TestStudio.GUI.ViewModels;
using System.IO;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Windows.Navigation;
using Unity;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using Telerik.Windows.Controls.Docking;
using System.Linq;
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;

namespace MerlinTestStudio_Demo_Telerik
{
/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : Window, IView
{
#region Private Member Variables

/// <summary>
/// Instance of this VIEWs VIEW-MODEL;
/// </summary>
//private MainWindowViewModel viewModelObj;

/// <summary>
/// Some images require a context for where they load images from. By setting the app base URI
all images can be loaded.
/// </summary>
//private Uri baseUri;

#endregion Private Member Variables

//Initialize Component
#region Constructor

public MainWindow()
{
var dataContext = new MainWindowViewModel();
dataContext.View = this as IView;
this.DataContext = dataContext;
```

```
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<MainWindowViewModel>();

//viewModelObj = (DataContext as MainWindowViewModel); //Set viewModelObj for window.
//DataStorage.MainViewModel = (DataContext as MainWindowViewModel); //Set MainViewModel for entire application reference.

// Set the ViewModels view property to be equal to this. This allows the viewModel access to the view but only
// through a controled interface thus not breaking the MVVM pattern.
//viewModelObj.View = this as IView;

InitializeComponent();

// Subscribe to the VIEW's loaded event and execute a method that in turn will execute a method in the View-Model when the View is loaded.
this.Loaded += new RoutedEventHandler(MainWindow_Loaded);

//EventManager.RegisterClassHandler(typeof(Window), Keyboard.KeyUpEvent, new KeyEventHandler(keyUp), true);

// Set the app's base URI.
//baseUri = BaseUriHelper.GetBaseUri(this);
#if !DEBUG
//Collapse the EZ access dev mode toggle.
DevModeEzAccessToggle.Visibility = Visibility.Collapsed;
#endif
}

#endregion

//Menu Item Click Events (Currently not in use).
#region Menu Bar Theme Events (Artifacts from 2019)

private void LightTheme_Click(object sender, RoutedEventArgs e)
{
//default color variation
VisualStudio2013Palette.LoadPreset(VisualStudio2013Palette.ColorVariation.Light);

// Set Non-Telerik controls colours.
MainWindowViewModel mwvm = this.DataContext as MainWindowViewModel;
```

```
if (mwvm != null)
{
mwvm.NonTelerikControlBackGroundColor = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#EEEEF2"));
}
}

private void DarkTheme_Click(object sender, RoutedEventArgs e)
{
//dark color variation
VisualStudio2013Palette.LoadPreset(VisualStudio2013Palette.ColorVariation.Dark);

// Set Non-Telerik controls colours.
MainWindowViewModel mwvm = this.DataContext as MainWindowViewModel;

if (mwvm != null)
{
mwvm.NonTelerikControlBackGroundColor = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#252526"));
}

}

#endregion

/// <summary>
/// Method that indicates to the view model that the view has been fully loaded.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
Style _style = null;

_style = (Style)Resources["CustomWindowStyle"];

this.Style = _style;

var viewModel = this.DataContext as MainWindowViewModel;
if (viewModel != null)
{
```

```
viewModel.LoadLayout();
}
}

/// <summary>
/// Closes the current window.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CloseClick(object sender, RoutedEventArgs e)
{
MainWindowViewModel mwvm = this.DataContext as MainWindowViewModel;

if ( mwvm != null)
{
mwvm.ExitProgramCommandExecute();
}

//var window = (Window)((FrameworkElement)sender).TemplatedParent;
//window.Close();
}

/// <summary>
/// Sets the Window size to be normal or maximized.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MaximizeRestoreClick(object sender, RoutedEventArgs e)
{
var window = (Window)((FrameworkElement)sender).TemplatedParent;
if (window.WindowState == System.Windows.WindowState.Normal)
{
window.WindowState = System.Windows.WindowState.Maximized;
}
else
{
window.WindowState = System.Windows.WindowState.Normal;
}
}

/// <summary>
/// Minimizes the current window.
```

```
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MinimizeClick(object sender, RoutedEventArgs e)
{
var window = (Window)((FrameworkElement)sender).TemplatedParent;
window.WindowState = System.Windows.WindowState.Minimized;
}


private void mainDocking_PreviewClose(object sender,
Telerik.Windows.Controls.Docking.StateChangeEventArgs e)
{
try
{
//int panesCount = DataStorage.MainViewModel.Panes.Count();
//RadPane rp = e.Panes.FirstOrDefault();
//if (rp != null)
//{
//    PVM pvm = (PVM)rp.DataContext;
//    if (pvm.IsDocument)
//    {
//        pvm.PaneClose();
//        //DataStorage.Panes.Remove(pvm);
//
//        panesCount = DataStorage.MainViewModel.Panes.Count();
//
//        //GC.Collect();
//    }
//}
//else { Console.WriteLine("RadPane on Preview Close is null..."); }

//////////////////////////////////////////////////////////////////////////////////

//// Get a reference to the mainwindowViewModel.
//MainWindowViewModel mwvm = this.DataContext as MainWindowViewModel;
//
//foreach (var pane in e.Panes)
//{
//    UserControl uc = pane.Content as UserControl;
//
//    if (uc != null)
```

```
//    {
//        UcViewModelBase viewModel = uc.DataContext as UcViewModelBase;
//
//        if (viewModel != null)
//        {
//            if (viewModel.SaveRequired == true)
//            {
//                string sMessageBoxText = "Save Changes to " +
Path.GetFileName(viewModel.FileName) + "?";
//                string sCaption = "Merlin Test Studio";
//
//                MessageBoxButton btnMessageBox = MessageBoxButton.YesNoCancel;
//                MessageBoxImage icnMessageBox = MessageBoxImage.Warning;
//
//                MessageBoxResult rsltMessageBox = MessageBox.Show(sMessageBoxText,
sCaption, btnMessageBox, icnMessageBox);
//
//                switch (rsltMessageBox)
//                {
//                    case MessageBoxResult.Yes:
//                        {
//                            viewModel.SaveDataFile();
//                            break;
//                        }
//                    case MessageBoxResult.No:
//                        {
//                            // User doesn't want to save changes to indicate a load is required.
//                            viewModel.IsDataLoaded = false;
//                            //viewModel.SaveRequired = false;
//
//                            if (mwvm != null)
//                            {
//                                mwvm.UpdateProjectTree(viewModel.FileName, false);
//                            }
//                        }
//                        break;
//
//                    case MessageBoxResult.Cancel:
//                        {
//                            e.Handled = true;
//                        }
//                        break;
```

```
//                }
//            }
//        }
//    }
//}
}
catch (Exception ex)
{
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}
}

private void RadPaneMaxMinButton_Click(object sender, RoutedEventArgs e)
{
var btn = sender as Button;
var group = btn.ParentOfType<RadPaneGroup>();
if (group != null)
{
var pane = group.SelectedPane;
if (!pane.IsFloating)
{
MaximizeExtensions.SetLastUsedGroup(pane, group);
pane.MakeFloatingDockable();
WindowCommands.Maximize.Execute(true, pane);

//go maximize
}
}
else
{
//// its toolwindow
var toolWindow = btn.ParentOfType<Telerik.Windows.Controls.Docking.ToolWindow>();
var splitcontainer = toolWindow.Content as RadSplitContainer;
var paneGroup = splitcontainer.Items[0] as RadPaneGroup;
var pane = paneGroup.SelectedPane;
var lastGroup = MaximizeExtensions.GetLastUsedGroup(pane);
if (lastGroup != null)
{
pane.RemoveFromParent();
lastGroup.Items.Add(pane);
}
else
```

```
{
if (toolWindow.WindowState == WindowState.Maximized)
{
WindowCommands.Restore.Execute(true, pane);
}
else
{
WindowCommands.Maximize.Execute(true, pane);
}
}
}
}

#region Docking Events & Methods
private void mainDocking_PreviewShowCompass(object sender,
Telerik.Windows.Controls.Docking.PreviewShowCompassEventArgs e)
{
bool isRootCompass = e.Compass is RootCompass;
var splitContainer = e.DraggedElement as RadSplitContainer;
if (splitContainer != null)
{
bool isDraggingDocument = splitContainer.EnumeratePanes().Any(p => p is RadDocumentPane);
var isTargetDocument = e.TargetGroup == null ? true : e.TargetGroup.EnumeratePanes().Any(p
=> p is RadDocumentPane);
if (isDraggingDocument)
{
e.Canceled = isRootCompass || !isTargetDocument;
}
else
{
e.Canceled = !isRootCompass && isTargetDocument;
}
}
}

private void mainDocking_Close(object sender,
Telerik.Windows.Controls.Docking.StateChangeEventArgs e)
{
var radPanes = (from p in e.Panes
where p.DataContext is PVM
where (p.DataContext as PVM).IsDocument
select p).ToList(); //Should get the Corresponding RadPanes.
```

```
var documentPVMs = e.Panes.Select(p => p.DataContext).OfType<PVM>().Where(vm =>
vm.IsDocument).ToList(); //Should get the corresponding PVMs.
for (int i = 0; i < radPanes.Count(); i++)
{
//Remove MTS Stuff.
documentPVMs[i].PaneClose();
DataStorage.MainViewModel.Panes.Remove(documentPVMs[i]);

//Nullify Telerik GUI stuff. //Already null when removing from collection.
//radPanes[i].Content = null;
//radPanes[i].Header = null;
//radPanes[i].DataContext = null;
//radPanes[i].RemoveFromParent();

///STILL NOT GC'ing correctly.
//Forcing a GC does not work.
//GC.Collect();
}
//foreach (PVM pvm in documentPVMs)
//{
//    pvm.PaneClose();
//    DataStorage.Panes.Remove(pvm);
//}
}

private void FilterActiveViewsSource(object sender, System.Windows.Data.FilterEventArgs e)
{
var vm = e.Item as PVM;
e.Accepted = vm.IsDocument;
}

private void FilterToolboxesSource(object sender, System.Windows.Data.FilterEventArgs e)
{
var vm = e.Item as PVM;
e.Accepted = !vm.IsDocument;
}

public RadDocking GetRadDocking()
{
return this.mainDocking;
}
```

```
#endregion

//Application Main Window Closing.
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
MainWindowViewModel mwvm = this.DataContext as MainWindowViewModel;

//Save the system configuration on close.
mwvm.MySystem.SaveToXml(BackendConstants.SystemFilePath);
mwvm.MyRecentData.SaveToXml(BackendConstants.RecentsDataFilePath);
}
}
}
```

CustomRadPane.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace Telerik.Windows.Controls
{
/// <summary>
/// Interaction logic for CustomRadPane.xaml
/// </summary>
public partial class CustomRadPane : UserControl
{
public CustomRadPane()
{
InitializeComponent();
```

```
}

public string FileName { get; set; }
}
}
```

DllFileInfo.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for DllFileInfo.xaml
/// </summary>
public partial class DllFileInfo : UserControl, IDLLInfoView
{
private DllFileInfoVM viewModelObj;

public DllFileInfo(DllFileInfoVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = dataContext;

((DllFileInfoVM)this.DataContext).View = this;
}
```

```
}
}
```

LicenceManager.xaml.cs

```
using System.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for LicenceManager.xaml
/// </summary>
public partial class LicenceManager : UserControl
{
public LicenceManager()
{
InitializeComponent();
}
}
}
```

PlaceHolderTextBox.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
```

```
/// Interaction logic for PlaceHolderTextBox.xaml
/// </summary>
public partial class PlaceHolderTextBox : UserControl
{
public PlaceHolderTextBox()
{
InitializeComponent();
DataContext = new PlaceHolderTextBoxVM();
}
}

public class PlaceHolderTextBoxVM : ViewModelBase, IPVMLink
{

private PaneViewModel _pvm;
public PaneViewModel PVM
{
get { return _pvm; }
set
{
_pvm = value;
OnPropertyChanged("PVM");
this.PlaceHolderText = (value.DataObject as WaveformModel).FilePath;
}
}

private string _placeHolderText = string.Empty;
public string PlaceHolderText
{
get { return _placeHolderText; } set { _placeHolderText = value;
OnPropertyChanged("PlaceHolderText"); }
}

}
}
```

Splash.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
```

```
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Reflection;

namespace MT.TestStudio.GUI
{
/// <summary>
/// Interaction logic for Splash.xaml
/// </summary>
public partial class Splash : Window
{
#region Private member variables.

private static Splash splash = new Splash();

// To refresh the UI immediately
private delegate void RefreshDelegate();

#endregion Private member variables.

#region Constructor

/// <summary>
/// Constructor for splash class.
/// </summary>
public Splash()
{
InitializeComponent();

DataContext = this;
}

#endregion Constructor

#region Methods.

/// <summary>
```

```
/// Loads and displays the splash immage.
/// </summary>
public static void BeginDisplay()
{
splash.Show();
}

/// <summary>
/// Stops displaying the splash screen.
/// </summary>
public static void EndDisplay()
{
// Update GUI elements on the UI thread based on what instruments are in the system.
if (splash.statuslbl.Dispatcher.CheckAccess() == true)
{
Console.WriteLine("splash.statuslbl.Dispatcher.CheckAccess() == true");
SafeCloseSplash();
}
else
{
Console.WriteLine("splash.statuslbl.Dispatcher.CheckAccess() == false");
//splash.statuslbl.Dispatcher.Invoke(new SafeCloseGUIDelegate(SafeCloseSplash), new object[] {
});
}
}

public static void Loading(string test)
{
if (splash.statuslbl.Dispatcher.CheckAccess() == true)
{
SafeUpdateSplashLabel(test);
}
else
{
splash.statuslbl.Dispatcher.Invoke(new SafeUpdateGUIDelegate(SafeUpdateSplashLabel), new
object[] { test });
}
}

private delegate void SafeUpdateGUIDelegate(string message);

private static void SafeUpdateSplashLabel(string message)
```

```
{
splash.statuslbl.Content = message;
Refresh(splash.statuslbl);
}

private delegate void SafeCloseGUIDelegate();

private static void SafeCloseSplash()
{
splash.Close();
}

public static void HideDisplay()
{
// Update GUI elements on the UI thread based on what instruments are in the system.
if (splash.statuslbl.Dispatcher.CheckAccess() == true)
{
SafeHideSplash();
}
else
{
splash.statuslbl.Dispatcher.Invoke(new SafeCloseGUIDelegate(SafeHideSplash), new object[] { });
}
}

private static void SafeHideSplash()
{
splash.Hide();
}

public string Version
{
get
{
return "Version " + Assembly.GetExecutingAssembly().GetName().Version.ToString();
}
}

/// <summary>
/// Refresh
/// </summary>
/// <param name="obj"></param>
```

```
private static void Refresh(DependencyObject obj)
{
obj.Dispatcher.Invoke(System.Windows.Threading.DispatcherPriority.Render,
(RefreshDelegate)delegate { });
}

#endregion Methods.
}
}
```

UserPreferences.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models.AppModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for UserPreferences.xaml
/// </summary>
public partial class UserPreferences : UserControl
{
public UserPreferences(UserPreferencesData dataContext)
{
this.DataContext = dataContext;

InitializeComponent();
}
```

```
private void SavePreferencesButton_Click(object sender, RoutedEventArgs e)
{
try
{
(this.DataContext as
UserPreferencesData).SaveToXml(BackendConstants.UserPreferencesDataFilePath);
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine(ex.Message);
}
finally
{
RadWindow window = this.ParentOfType<RadWindow>();
window.Close();
}

}
}
}
```

WindowStyled.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Threading;

namespace MT.TestStudio.GUI
{
```

```
/// <summary>
/// Interaction logic for WindowStyled.xaml
/// </summary>
public partial class WindowStyled : Window
{
public WindowStyled(MainWindowViewModel MainWindowViewModel)
{
InitializeComponent();

// this.Resources.MergedDictionaries.Add(MainWindowViewModel.LanguageDict[0]);

// Subscribe to the VIEW's loaded event.
this.Loaded += new RoutedEventHandler(MainWindow_Loaded);
}

/// <summary>
/// Method that indicates to the view model that the view has been fully loaded.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
// To get the windows chrome ( top title bar ) to be styled we need
// to override the current style.
Style _style = null;

_style = (Style)Resources["CustomWindowStyle"];

this.Style = _style;
}

#region Public Methods

/// <summary>
/// Method that causes the window to close.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CloseClick(object sender, RoutedEventArgs e)
{
var window = (Window)((FrameworkElement)sender).TemplatedParent;
window.Close();
```

```
}

/// <summary>
/// Method that causes the window to be restored.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void MaximizeRestoreClick(object sender, RoutedEventArgs e)
{
var window = (Window)((FrameworkElement)sender).TemplatedParent;
if (window.WindowState == System.Windows.WindowState.Normal)
{
window.WindowState = System.Windows.WindowState.Maximized;
}
else
{
window.WindowState = System.Windows.WindowState.Normal;
}
}

private void MinimizeClick(object sender, RoutedEventArgs e)
{
var window = (Window)((FrameworkElement)sender).TemplatedParent;
window.WindowState = System.Windows.WindowState.Minimized;
}

/// <summary>
/// Method that displays a dialog box.
/// </summary>
/// <param name="message"></param>
public void ShowDialog(string message)
{

// Check to see if we're on the UI thread before showing the dialog box.
if (Application.Current.Dispatcher.CheckAccess())
{
MessageBox.Show(Application.Current.MainWindow, message, "PXI Amplifier Test
Demonstration");
}
else
{
Application.Current.Dispatcher.Invoke(DispatcherPriority.Normal, new Action(() =>
```

```
{
MessageBox.Show(Application.Current.MainWindow, message, "PXI Amplifier Test
Demonstration");
}));
}
}

#endregion Public Methods
}
}
```

AttenuatorResultsViewer.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Microsoft.Win32;
using System;
using System.IO;
using System.Windows;
using System.Windows.Controls;
using Telerik.Charting;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for AttenuatorResultsViewer.xaml
/// </summary>
public partial class AttenuatorResultsViewer : UserControl
{
public AttenuatorResultsViewer(AttenuatorResultsViewerVM dataContext)
{
InitializeComponent();
//var result = (this.DataContext as PaneViewModel).PassedCalObject;
//this.DataContext = new AttenuatorResultsViewerVM(result as CalResult);
this.DataContext = dataContext;
}

/// <summary>
/// When you click on a data point in the grid view, the trackball moves to that point relative to the
chart.
/// </summary>
/// <param name="sender"></param>
```

```
/// <param name="e"></param>
private void RadGridView_SelectionChanged(object sender,
Telerik.Windows.Controls.SelectionChangeEventArgs e)
{
KvPOfDoubleDouble selected_kv = ChartDataGridView.SelectedItem as KvPOfDoubleDouble;
if ((bool)Y_Delta_Switch.IsChecked)
{
trackballBehavior.Position = Chart.ConvertDataToPoint(new DataTuple(selected_kv.Key,
selected_kv.Delta < 0 ? selected_kv.Delta * -1 : selected_kv.Delta));
}
else
{
trackballBehavior.Position = Chart.ConvertDataToPoint(new DataTuple(selected_kv.Key,
selected_kv.Value < 0 ? selected_kv.Value * -1 : selected_kv.Value));
}
}

/// <summary>
/// Export event for the gridview data.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RadButton_Click(object sender, RoutedEventArgs e)
{
var vm = this.DataContext as AttenuatorResultsViewerVM;
string extension = "csv";
SaveFileDialog dialog = new SaveFileDialog()
{
DefaultExt = extension,
Filter = String.Format("{1} files (.{0})|.{0}|All files (.)|.", extension, "Excel"),
FilterIndex = 1,
FileName = string.Format("{0}.{1}-{2}-{3}-{4}", vm.resultParentName, vm.resultName,
vm.SelectedFrequency, vm.SelectedAmpConfig, vm.SelectedGroup)
};
if (dialog.ShowDialog() == true)
{
using (Stream stream = dialog.OpenFile())
{
ChartDataGridView.Export(stream,
new GridViewExportOptions()
{
Format = ExportFormat.Csv,
```

```
ShowColumnHeaders = true,
ShowColumnFooters = true,
ShowGroupFooters = false,
});
}
}
}
}
}
```

CalDataInfo.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Windows;
using System.Windows.Controls;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for CalDataInfo.xaml
/// </summary>
public partial class CalDataInfo : UserControl
{
//Constructor for Previews window creation.
public CalDataInfo()
{
InitializeComponent();
}
//Constructor for Pane Creation.
public CalDataInfo(ResultDataVM dataContext)
{
InitializeComponent();
//var cdm = (this.DataContext as PaneViewModel).PassedCalObject;
this.DataContext = dataContext.CalDataInfo;
}

private void CalPointsGridView_AutoGeneratingColumn(object sender,
Telerik.Windows.Controls.GridViewAutoGeneratingColumnEventArgs e)
{
var dataColumn = e.Column as GridViewDataColumn;
if (dataColumn != null)
```

```
{
//Boolean columns
if (e.ItemPropertyInfo.PropertyType == typeof(bool) || e.ItemPropertyInfo.PropertyType ==
typeof(bool?))
{
GridViewCheckBoxColumn newColumn = new GridViewCheckBoxColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;
newColumn.Header = dataColumn.Header;
newColumn.UniqueName = dataColumn.UniqueName;
e.Column = newColumn;
}

//Double columns
if (e.ItemPropertyInfo.PropertyType == typeof(double))
{
if (e.Column.UniqueName == "Frequency" || e.Column.UniqueName == "Key")
{
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;
newColumn.Header = "Frequency";
newColumn.UniqueName = dataColumn.UniqueName;
e.Column = newColumn;
}
else
{
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;
newColumn.Header = dataColumn.Header;
newColumn.UniqueName = dataColumn.UniqueName;
newColumn.DataFormatString = "{0:N3}";
e.Column = newColumn;
}
}

//if (e.Column.UniqueName == "Value")
//{
//    e.Cancel = true;
//}
}
}

}
```

}

CalibrationConfiguratorView.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Unity;
using System.IO;
using Telerik.Windows.Data;
using MerlinTestStudio_Demo_Telerik.Data;
using System.Reflection;
using System.Windows.Input;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for CalibrationConfiguratorView.xaml
/// </summary>
public partial class CalibrationConfiguratorView : UserControl, MainView
{
private CalibrationConfiguratorVM viewModelObj;
private ObservableCollection<ContextMenuItem> rowContextMenuItems;
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

public CalibrationConfiguratorView(CalibrationConfiguratorVM dataContext)
{
InitializeComponent();
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<CalibrationConfiguratorVM>();
//OLD
this.DataContext = dataContext;
```

```
viewModelObj = dataContext;

viewModelObj.View = this as MainView;

//Enables row reorder behavior for the specified Gridview.
RowReorderBehavior.SetIsEnabled(CalPointGridView, true);
CalPointGridView.Drop += DropChanges;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.CalPointGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.CalPointGridView);

//Context menu items for CalPoint editor
ObservableCollection<ContextMenuItem> groupItems = new
ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Insert" },
new ContextMenuItem() { Text = "Delete" },
new ContextMenuItem() { Text = "Duplicate" }
};
this.rowContextMenuItems = groupItems;
}

private void DropChanges(object sender, DragEventArgs e)
{
viewModelObj.CurrentCalConfig.OnChangeOccured();
}

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
if (ClickedRow != null && ClickedRow.IsInEditMode == false) //ClickedRow.IsInEditMode == false;
Prevents major bug!
{
foreach (var item in this.rowContextMenuItems) { item.IsEnabled = true; }
GridContextMenu.ItemsSource = rowContextMenuItems;
if(!CalPointGridView.SelectedItems.Contains(ClickedRow.DataContext as CalPointModel))
{
CalPointGridView.SelectedCells.Clear();
CalPointGridView.SelectedItem = ClickedRow.DataContext as CalPointModel;
}
}
else
```

```
{
foreach (var item in this.rowContextMenuItems)
{
switch(item.Text)
{
case "Insert": item.IsEnabled = true; break;
case "Delete": item.IsEnabled = false; break;
case "Duplicate": item.IsEnabled = false; break;
}
}
GridContextMenu.ItemsSource = rowContextMenuItems;
}
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
switch (item.Text)
{
case "Insert": viewModelObj.InsertCalPointCommand.Execute(CalPointGridView.SelectedItems);
break;
case "Delete":
viewModelObj.RemoveCalPointsCommand.Execute(CalPointGridView.SelectedItems); break;
case "Duplicate":
viewModelObj.DuplicateCalPointCommand.Execute(CalPointGridView.SelectedItems); break;
}
}

/// <summary>
/// Utilizing the cell edit ended event to acheive multi-edit capability
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CalPointGridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
try
{
if (e.EditAction == GridViewEditAction.Commit) //&&  e.NewData != ""
{
string FirstPropertyChanged = e.Cell.Column.UniqueName;

List<Edit<int, object, string>> editedCalPoints = new List<Edit<int, object, string>>();
```

```
//Log the old value(s) and set the new value(s) of each CalPoint Selected.
for (int i = 0; i < CalPointGridView.SelectedCells.Count; i++)
{
GridViewCellInfo cell = CalPointGridView.SelectedCells[i];
object calPointModelObj = cell.Item;
string cellPropertyName = cell.Column.UniqueName;
object OldValue;

PropertyInfo firstProp = calPointModelObj.GetType().GetProperty(FirstPropertyChanged);
PropertyInfo cellProp = calPointModelObj.GetType().GetProperty(cellPropertyName);

if (firstProp.PropertyType == cellProp.PropertyType) //Protects against cross-selection type
conversion errors.
{
OldValue = cellProp.GetValue(calPointModelObj); //Get old value before it's overwrote.
if (i > 0) //Skip the first cell as the grid already commits the data.
{
cellProp.SetValue(calPointModelObj, e.NewData);
}

//This fixes a problem with the item ending the edit overwriting the old value with the new one.
if (calPointModelObj == e.Cell.ParentRow.Item) { OldValue = e.OldData; }

//Logs the index of the CalPoint edited and it's previous value.
editedCalPoints.Add(new Edit<int, object,
string>(CalPointGridView.Items.IndexOf(calPointModelObj), OldValue, cellPropertyName));
}
}

viewModelObj.CurrentCalConfig.LogEditAction(editedCalPoints);

//Forces OnChangesMade when the user changes the edits a CalPoint.
viewModelObj.CurrentCalConfig.OnChangeOccured();
}
}
catch(Exception ex) { MessageBox.Show(ex.Message); }
}

/// <summary>
/// Fires when the user hits the Delete key. This re-routes the action to the ViewModel so it can be
properly logged.
/// </summary>
```

```
/// <param name="sender"></param>
/// <param name="e"></param>
private void CalPointGridView_Deleting(object sender, GridViewDeletingEventArgs e)
{
viewModelObj.RemoveCalPointsCommand.Execute(CalPointGridView.SelectedItems);
e.Cancel = true;
}

#region MainView interface implementation
/// <summary>
/// Unselects CalPoint(s) when the user hits Esc key.
/// </summary>
public void UnselectCalPoints()
{
this.CalPointGridView.UnselectAll();
///try
///{
///    var scope = System.Windows.Input.FocusManager.GetFocusScope(CalPointGridView);
///    System.Windows.Input.FocusManager.SetFocusedElement(scope, null);
///    System.Windows.Input.Keyboard.ClearFocus();
///}
///catch (Exception ex) { MessageBox.Show(ex.Message); }
}

/// <summary>
/// Force the GridView to commit any pending edits.
/// </summary>
public void ForceCommitEdit()
{
this.CalPointGridView.CommitEdit();
}

/// <summary>
/// Used for undo functionality purposes. May not be needed.
/// </summary>
public void GridViewInvalidateMeasure()
{
this.CalPointGridView.ChildrenOfType<ScrollViewer>().First().InvalidateMeasure();
}
#endregion

//Fires per selected cell that the COPY action was performed on.
```

```
private void CalPointGridView_PastingCellClipboardContent(object sender,
GridViewCellClipboardEventArgs e)
{
try
{
List<Edit<int, object, string>> editedCalPoints = new List<Edit<int, object, string>>();
object calPointModelObj = e.Cell.Item;
string cellPropertyName = e.Cell.Column.UniqueName;

//Logs the index of the CalPoint edited and it's previous value.
object OldValue =
calPointModelObj.GetType().GetProperty(cellPropertyName).GetValue(calPointModelObj);
editedCalPoints.Add(new Edit<int, object,
string>(CalPointGridView.Items.IndexOf(calPointModelObj), OldValue, cellPropertyName));
viewModelObj.CurrentCalConfig.LogEditAction(editedCalPoints);

//Forces OnChangesMade when the user changes the edits a CalPoint.
viewModelObj.CurrentCalConfig.OnChangeOccured();
}
catch(Exception ex) { MessageBox.Show(ex.Message); };
}



private void CalPointGridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//This stops the user from selecting rows and accidently making a multi-edit on all the selected
cells within, also while preserving action logging for undo capability.
if (CalPointGridView.SelectedItems.Count() > 0)
{
var cell = CalPointGridView.CurrentCell;
CalPointGridView.SelectedItems.Clear();
CalPointGridView.SelectedCells.Add(new GridViewCellInfo(cell));
}
}

private void CalPointGridView_Loaded(object sender, RoutedEventArgs e)
{
//IsSynchronizedWithCurrentItem ="True", set CurrentItem to null so the first row isn't selected on
on load
CalPointGridView.CurrentItem = null;
}
```

```
private void UserControl_Unloaded(object sender, RoutedEventArgs e)
{
//viewModelObj = null; //Called when the user switched active panes.
}

}

#region ContextMenu MenuItemObjects

public class ContextMenuItem : INotifyPropertyChanged
{
private bool isEnabled = true;
private bool _isSeparator = false;
private string text;
private ObservableCollection<ContextMenuItem> subItems;

//Constructor should require a command to initialize an object. (if this is to become the standard for making context menus.
public ContextMenuItem()
{

}


public ICommand Command { get; set; }
//public object CommandParameter { get; set; }

public bool IsEnabled
{
get { return this.isEnabled; }
set
{
if (this.isEnabled != value)
{
this.isEnabled = value;
this.OnNotifyPropertyChanged("IsEnabled");
}
}
}
public bool IsSeparator
{
```

```
get { return this._isSeparator; }
set
{
if (this._isSeparator != value)
{
this._isSeparator = value;
this.OnNotifyPropertyChanged("IsSeparator");
}
}
}
public string Text
{
get { return this.text; }
set
{
if (this.text != value)
{
this.text = value;
this.OnNotifyPropertyChanged("Text");
}
}
}
public ObservableCollection<ContextMenuItem> SubItems
{
get
{
if (this.subItems == null)
{
this.subItems = new ObservableCollection<ContextMenuItem>();
}
return this.subItems;
}
set
{
if (this.subItems != value)
{
this.subItems = value;
this.OnNotifyPropertyChanged("SubItems");
}
}
}
```

```
#region INPC Members

public event PropertyChangedEventHandler PropertyChanged;
private void OnNotifyPropertyChanged(string ptopertyName)
{
if (this.PropertyChanged != null)
{
this.PropertyChanged(this, new PropertyChangedEventArgs(ptopertyName));
}
}

#endregion
}

#endregion
}
```

LimitUcEditor.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Converters;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.CalibrationViewModels;
using System;
using System.Collections.Generic;
using System.Dynamic;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Calibration
{
```

```
/// <summary>
/// Interaction logic for ProgramTestLimitEditor.xaml
/// </summary>
public partial class LimitUcEditor : UserControl, ILimitUcView
{
private LimitUcVM viewModelObj;

public LimitUcEditor(LimitUcVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = DataContext as LimitUcVM;

viewModelObj.View = this;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
//CAUSES BUG: Multi-Edit bug where the selected cell is the one being edited where as the move down cell is taking the new data... need fix to comply with current grid standards.
///this.LimitEditorGridView.KeyboardCommandProvider = new CustomKeyboardCommandProvider(this.LimitEditorGridView);


// this.LimitEditorGridView.RightFrozenColumnCount = 5;
this.LimitEditorGridView.RightFrozenColumnsSplitterVisibility = Visibility.Visible;
}


private void GridView_AutoGeneratingColumn(object sender, GridViewAutoGeneratingColumnEventArgs e)
{
var dataColumn = e.Column as GridViewDataColumn;

//Column nane specific, not type specific.
if (dataColumn != null)
{
string headerText;
Style CalStepCellStyle = Application.Current.FindResource("CalStepCellStyle") as Style;
Style PassFailCellStyle = Application.Current.FindResource("PassFailCellStyle") as Style;
Style AmplifiersCompressedCellStyle = Application.Current.FindResource("AmplifiersCompressedCellStyle") as Style;
```

```
switch (e.Column.UniqueName)
{
case "WasTargetLevelAcheived":
headerText = "Cal Step"; //headerText = "Pass/Fail";
dataColumn.DataMemberBinding.Converter = new BooleanToPassFailConverter();
e.Column = new GridViewDataColumn()
{
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None,
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
CellStyle = CalStepCellStyle
};
break;
case "AmplifiersCompressed": //Make sure name is correct.
headerText = "Amplifiers"; //headerText = "Amplifiers Compressed";
dataColumn.DataMemberBinding.Converter = new InverseBooleanToPassFailConverter();
e.Column = new GridViewDataColumn()
{
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None,
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
CellStyle = AmplifiersCompressedCellStyle
};
break;
case "DcAmplifierEnabled": //apart of Key object.
case "dcRfAmpEnabled": //apart of Value object.
headerText = dataColumn.Header.ToString();
dataColumn.DataMemberBinding.Converter = new BooleanToYesNoConverter();
e.Column = new GridViewDataColumn()
{
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None,
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
//CellStyle = PassFailCellStyle
};
break;
case "Frequency":
case "Key": //Same as below but with slightly different header (fast fix).
```

```
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None;
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;

#region Frequency Converter
//Set the converter for the column.
newColumn.DataMemberBinding.Converter = new FrequencyConverter();
newColumn.DataMemberBinding.ConverterParameter =
this.ViewingUnitComboBox.SelectedValue;
//newColumn.DataFormatString = "{0} " + this.ViewingUnitComboBox.SelectedValue; //Shows the
Unit right next to the data in each cell.
#endregion

newColumn.Header = "Frequency";
newColumn.UniqueName = dataColumn.UniqueName;
e.Column = newColumn;
break;
case "ifFreq":
GridViewDataColumn newIfFreqColumn = new GridViewDataColumn();
newIfFreqColumn.EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None;
newIfFreqColumn.DataMemberBinding = dataColumn.DataMemberBinding;

#region Frequency Converter
//Set the converter for the column.
newIfFreqColumn.DataMemberBinding.Converter = new FrequencyConverter();

//"{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:ResultDataView}}, Path=DataContext.ViewingUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
//var binding = new Binding("ViewingUnit")
//{
//    Mode = BindingMode.OneWay,
//    UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
//    RelativeSource = new RelativeSource(RelativeSourceMode.FindAncestor,
typeof(ResultDataView), 1),
//    Path = new PropertyPath("DataContext.ViewingUnit")
//};

//WORK AROUND. Needs new column generation as soon as SelectedValue is Changed, bad for
GUI, makes it slower.
newIfFreqColumn.DataMemberBinding.ConverterParameter =
this.ViewingUnitComboBox.SelectedValue;
```

```
newIfFreqColumn.DataFormatString = "{0} " + this.ViewingUnitComboBox.SelectedValue;
#endregion

newIfFreqColumn.Header = "If Frequency";
newIfFreqColumn.UniqueName = dataColumn.UniqueName;
e.Column = newIfFreqColumn;
break;
case "Min":
case "Max":
case "MinTolerance":
case "MaxTolerance":
case "Baseline":
//Create Columns w/ edit triggers.
GridViewDataColumn newColumn1 = new GridViewDataColumn() {  };
dataColumn.DataMemberBinding.Mode = BindingMode.TwoWay; //Bind Edit.
dataColumn.DataMemberBinding.UpdateSourceTrigger =
UpdateSourceTrigger.PropertyChanged; //Bind Edit.
newColumn1.DataMemberBinding = dataColumn.DataMemberBinding;
if(e.Column.UniqueName == "Baseline") { newColumn1.EditTriggers =
Telerik.Windows.Controls.GridView.GridViewEditTriggers.None; }
newColumn1.Header = dataColumn.Header;
newColumn1.UniqueName = dataColumn.UniqueName;
//if (e.Column.UniqueName == "Min" || e.Column.UniqueName == "Max") {
newColumn1.DataFormatString = "{0:N3}"; } //Visually fixes a calulation bug that produces too
many decimal places.
e.Column = newColumn1;
break;
default:
GridViewDataColumn newColumn2 = new GridViewDataColumn();
newColumn2.DataMemberBinding = dataColumn.DataMemberBinding;
newColumn2.EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None;
///switch (dataColumn.UniqueName)
///{
///    case "GainCPLBoost": newColumn.Header = "RF110 A1"; break;
///    case "GainLnaA": newColumn.Header = "RF110 A2"; break;
///    case "GainLnaB": newColumn.Header = "RF110 A3"; break;
///    case "GainA1": newColumn.Header = "RF210 A1"; break;
///    case "GainA2": newColumn.Header = "RF210 A2"; break;
///    case "GainA3": newColumn.Header = "RF210 A3"; break;
///}
newColumn2.Header = dataColumn.Header;
```

```
newColumn2.UniqueName = dataColumn.UniqueName;
//newColumn2.DataFormatString = "{0:N3}";
e.Column = newColumn2;
///Previous default Bool Column code.
//headerText = dataColumn.Header.ToString();
//e.Column = new GridViewCheckBoxColumn()
//{
//    EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.None,
//    DataMemberBinding = dataColumn.DataMemberBinding,
//    Header = headerText,
//    UniqueName = dataColumn.UniqueName,
//    CellStyle = PassFailCellStyle
//};
break;
}

if (e.Column.UniqueName == "Value")
{
e.Cancel = true;
}
}
}

//WORK AROUND
//On viewing unit selection changed, update the autogenerated columns by forcing column
generation to update
private void ViewingUnitComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
this.LimitEditorGridView.AutoGenerateColumns = false;
this.LimitEditorGridView.AutoGenerateColumns = true;
}

public void ForceCommitEdit()
{
this.LimitEditorGridView.CommitEdit();
}

//May not be needed.
public void RebindGridView()
{
this.LimitEditorGridView.Rebind();
```

```
}

private void LimitEditorGridView_CellValidating(object sender, GridViewCellValidatingEventArgs
e)
{
try
{
double newDataParsed = double.Parse(e.NewValue.ToString());
dynamic DynExObj = e.Row.Item;
double Baseline = (double)DynExObj.Baseline;

switch (e.Cell.Column.UniqueName)
{
case "Min":
if (newDataParsed >= Baseline)
{
e.IsValid = false;
e.ErrorMessage = "Min cannot be greater than or equal to the Baseline.";
}
else { e.IsValid = true; }
break;
case "Max":
if (newDataParsed <= Baseline)
{
e.IsValid = false;
e.ErrorMessage = "Max cannot be less than or equal to the Baseline.";
}
else { e.IsValid = true; }
break;
case "MinTolerance":
case "MaxTolerance":
if (newDataParsed < 0) //Check for negative value.
{
e.IsValid = false;
e.ErrorMessage = "Tolerance cannot be set as a negative value.";
}
else { e.IsValid = true; }
break;
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
```

```
}

private void LimitEditorGridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs
e)
{
try
{
if (e.EditAction == GridViewEditAction.Commit) //&&  e.NewData != ""
{
//NaN Check, don't allow NaN as an input.
if(e.NewData is double && double.IsNaN((double)e.NewData) ) //TEST
{
Console.WriteLine("User input 'NaN' is not allowed.");
//e.Handled = true;
return; //Stop the code, do not commit edit to object
}

for (int i = 0; i < this.LimitEditorGridView.SelectedCells.Count; i++)
{
GridViewCellInfo cell = this.LimitEditorGridView.SelectedCells[i];

dynamic DynExObj = cell.Item;
CalculateDynamicExpandoObjectProperties(ref DynExObj, cell.Column.UniqueName,
e.NewData);
}

viewModelObj.CalLimits.OnChangeOccured();
}
}
catch (Exception ex) { MessageBox.Show(ex.Message);  Console.WriteLine(ex.Message); }
}

private void LimitEditorGridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//This stops the user from selecting rows and accidently making a multi-edit on all the selected
cells within (also while preserving action logging for undo capability.)
if (this.LimitEditorGridView.SelectedItems.Count() > 0)
{
var cell = this.LimitEditorGridView.CurrentCell;
this.LimitEditorGridView.SelectedItems.Clear();
this.LimitEditorGridView.SelectedCells.Add(new GridViewCellInfo(cell));
```

```
}
}

//Fires per selected cell that the paste action is performed on.
private void LimitEditorGridView_PastingCellClipboardContent(object sender,
GridViewCellClipboardEventArgs e)
{
try
{
dynamic DynExObj = e.Cell.Item;
CalculateDynamicExpandoObjectProperties(ref DynExObj, e.Cell.Column.UniqueName, e.Value);

viewModelObj.CalLimits.OnChangeOccured();
}
catch (Exception ex) { MessageBox.Show(ex.Message); };
}

private void CalculateDynamicExpandoObjectProperties(ref dynamic DynExObj, string
propertyName, object NewData)
{
double newDataParsed = double.Parse(NewData.ToString());
double Baseline = (double)DynExObj.Baseline;

switch (propertyName)
{
case "Min":
double minTolNan = double.Parse(DynExObj.MinTolerance.ToString());
if (!Double.IsNaN(minTolNan))
{
DynExObj.Min = newDataParsed;
DynExObj.MinTolerance = Math.Abs(Baseline - DynExObj.Min);
}
else { DynExObj.Min = double.NaN; }
break;
case "Max":
double maxTolNaN = double.Parse(DynExObj.MaxTolerance.ToString());
if (!Double.IsNaN(maxTolNaN))
{
DynExObj.Max = newDataParsed;
DynExObj.MaxTolerance = Math.Abs(DynExObj.Max - Baseline);
}
else { DynExObj.Max = double.NaN; }
```

```
break;
case "MinTolerance":
double minNaN = double.Parse(DynExObj.Min.ToString());
if (!Double.IsNaN(minNaN))
{
DynExObj.MinTolerance = Math.Abs(newDataParsed);
DynExObj.Min = Baseline - DynExObj.MinTolerance;
}
else { DynExObj.MinTolerance = double.NaN; }
break;
case "MaxTolerance":
double maxNaN = double.Parse(DynExObj.Max.ToString());
if (!Double.IsNaN(maxNaN))
{
DynExObj.MaxTolerance = Math.Abs(newDataParsed);
DynExObj.Max = Baseline + DynExObj.MaxTolerance;
}
else { DynExObj.MaxTolerance = double.NaN; }
break;
}

}

}
}
```

ResultDataView.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Converters;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
```

```
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for ResultDataView.xaml
/// </summary>
public partial class ResultDataView : UserControl
{
public ResultDataView(ResultDataVM dataContext)
{
InitializeComponent();
//var result = (this.DataContext as PaneViewModel).PassedCalObject;
//this.DataContext = new ResultDataVM(result as CalResult);
this.DataContext = dataContext;
}

/// <summary>
/// Export event for the gridview data.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RadMenuItem_Click(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
var vm = this.DataContext as ResultDataVM;
string extension = "csv";
SaveFileDialog dialog = new SaveFileDialog()
{
DefaultExt = extension,
Filter = String.Format("{1} files (.{0})|.{0}|All files (.)|.", extension, "Excel"),
FilterIndex = 1,
FileName = string.Format("{0}.{1}", vm.resultParentName, vm.resultName)
};
if (dialog.ShowDialog() == true)
{
```

```
//before your loop
var csv = new StringBuilder();

foreach(var resultItem in vm.GridViewData)
{
string lineToAppend = string.Empty;
foreach(PropertyInfo prop in resultItem.GetType().GetProperties())
{
lineToAppend += prop.GetValue(resultItem);
lineToAppend += ",";
}
csv.AppendLine(lineToAppend);
}

//after your loop
File.WriteAllText(dialog.FileName, csv.ToString());

}
}
catch(Exception ex ) { MessageBox.Show(ex.Message); }
}

private void CalResultGridView_AutoGeneratingColumn(object sender,
GridViewAutoGeneratingColumnEventArgs e)
{
var dataColumn = e.Column as GridViewDataColumn;
if (dataColumn != null)
{
//Boolean columns
if (e.ItemPropertyInfo.PropertyType == typeof(bool) || e.ItemPropertyInfo.PropertyType ==
typeof(bool?))
{
string headerText;
Style CalStepCellStyle = Application.Current.FindResource("CalStepCellStyle") as Style;
Style PassFailCellStyle = Application.Current.FindResource("PassFailCellStyle") as Style;
Style AmplifiersCompressedCellStyle =
Application.Current.FindResource("AmplifiersCompressedCellStyle") as Style;

switch (e.Column.UniqueName)
{
case "WasTargetLevelAcheived":
headerText = "Cal Step"; //headerText = "Pass/Fail";
```

```
dataColumn.DataMemberBinding.Converter = new BooleanToPassFailConverter();
e.Column = new GridViewDataColumn()
{
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
CellStyle = CalStepCellStyle
};
break;
case "AmplifiersCompressed": //Make sure name is correct.
headerText = "Amplifiers"; //headerText = "Amplifiers Compressed";
dataColumn.DataMemberBinding.Converter = new InverseBooleanToPassFailConverter();
e.Column = new GridViewDataColumn()
{
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
CellStyle = AmplifiersCompressedCellStyle
};
break;
case "PassLimitStatus":
case "LimitPassStatus":
//Change on HOLD.
//Some how transfer value to CalFactor (if applicable) for red hightlight (if failed)
e.Cancel = true; //Don't generate this column.
break;
case "ResultPass":
headerText = dataColumn.Header.ToString();
dataColumn.DataMemberBinding.Converter = new BooleanToYesNoConverter();
e.Column = new GridViewDataColumn()
{
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
//CellStyle = PassFailCellStyle
};
break;
case "DcAmplifierEnabled": //apart of Key object.
case "dcRfAmpEnabled": //apart of Value object.
headerText = dataColumn.Header.ToString();
dataColumn.DataMemberBinding.Converter = new BooleanToYesNoConverter();
e.Column = new GridViewDataColumn()
```

```
{
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
//CellStyle = PassFailCellStyle
};
break;
default: //DEFAULT
headerText = dataColumn.Header.ToString();
e.Column = new GridViewCheckBoxColumn()
{
DataMemberBinding = dataColumn.DataMemberBinding,
Header = headerText,
UniqueName = dataColumn.UniqueName,
CellStyle = PassFailCellStyle
};
break;
}

}

#region Enum Column Comments
///Enum type fields get generated as GridViewDataColumn(s).
//if (e.ItemPropertyInfo.PropertyType == typeof(Enum))
//{
//    GridViewDataColumn newColumn = new GridViewDataColumn();
//    newColumn.DataMemberBinding = dataColumn.DataMemberBinding;
//    newColumn.Header = "Frequency";
//    newColumn.UniqueName = dataColumn.UniqueName;
//    e.Column = newColumn;
//}
#endregion

//Double columns
if (e.ItemPropertyInfo.PropertyType == typeof(double))
{
if (e.Column.UniqueName == "Frequency" || e.Column.UniqueName == "Key")
{
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;

#region Frequency Converter
```

```
//Set the converter for the column.
newColumn.DataMemberBinding.Converter = new FrequencyConverter();

//"{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type
local:ResultDataView}}, Path=DataContext.ViewingUnit, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"
//var binding = new Binding("ViewingUnit")
//{
//    Mode = BindingMode.OneWay,
//    UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged,
//    RelativeSource = new RelativeSource(RelativeSourceMode.FindAncestor,
typeof(ResultDataView), 1),
//    Path = new PropertyPath("DataContext.ViewingUnit")
//};

//WORK AROUND. Needs new column generation as soon as SelectedValue is Changed, bad for
GUI, makes it slower.
newColumn.DataMemberBinding.ConverterParameter =
this.ViewingUnitComboBox.SelectedValue;

//newColumn.DataFormatString = "{0} " + this.ViewingUnitComboBox.SelectedValue; //Shows the
Unit right next to the data in each cell.
#endregion

newColumn.Header = "Frequency";
newColumn.UniqueName = dataColumn.UniqueName;
e.Column = newColumn;
}
else if(e.Column.UniqueName == "ifFreq")
{
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;

//Same code as above...
#region Frequency Converter
//Set the converter for the column.
newColumn.DataMemberBinding.Converter = new FrequencyConverter();
newColumn.DataMemberBinding.ConverterParameter =
this.ViewingUnitComboBox.SelectedValue;
newColumn.DataFormatString = "{0} " + this.ViewingUnitComboBox.SelectedValue;
#endregion
```

```
newColumn.Header = "If Frequency";
newColumn.UniqueName = dataColumn.UniqueName;
e.Column = newColumn;
}
else
{
GridViewDataColumn newColumn = new GridViewDataColumn();
newColumn.DataMemberBinding = dataColumn.DataMemberBinding;
switch (dataColumn.UniqueName)
{
case "GainCPLBoost": newColumn.Header = "RF110 A1"; break;
case "GainLnaA": newColumn.Header = "RF110 A2"; break;
case "GainLnaB": newColumn.Header = "RF110 A3"; break;
case "GainA1": newColumn.Header = "RF210 A1"; break;
case "GainA2": newColumn.Header = "RF210 A2"; break;
case "GainA3": newColumn.Header = "RF210 A3"; break;
}
//newColumn.Header = dataColumn.Header;
newColumn.UniqueName = dataColumn.UniqueName;
newColumn.DataFormatString = "{0:N3}";
e.Column = newColumn;
}
}

if (e.Column.UniqueName == "Value")
{
e.Cancel = true; //Don't generate this column.
}
}
}

//WORK AROUND
//On viewing unit selection changed, update the autogenerated columns by forcing column
generation to update
private void ViewingUnitComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
this.CalResultGridView.AutoGenerateColumns = false;
this.CalResultGridView.AutoGenerateColumns = true;
}
}
}
```

ProductConfigurationView.xaml.cs

```
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for ProductConfiguration.xaml
/// </summary>
public partial class ProductConfigurationView : UserControl, IProductConfigView
{
private ProductConfigViewModel viewModelObj;

public ProductConfigurationView(ProductConfigViewModel dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = dataContext;

((ProductConfigViewModel)this.DataContext).View = this;
}



}
}
```

ProjectConfiguration.xaml.cs

```
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Unity;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for ProjectConfiguration.xaml
/// </summary>
public partial class ProjectConfiguration : UserControl
{
public ProjectConfiguration()
{
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<ProjectConfigViewModel>();

this.DataContext = new ProjectConfigViewModel();

InitializeComponent();

}

private void RadComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
if(this.IsLoaded)
{
Console.WriteLine("Project target system has been changed, press save all to save changes...");
}
//if (e.AddedItems.Count > 0)
```

```
//{
//    Console.WriteLine("Project target system has been changed, press save all to save
changes...");
//}
}
}
}
```

SystemConfiguration.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Windows;
using System.Windows.Controls;
using Unity;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for SystemConfiguration.xaml
/// </summary>
public partial class SystemConfiguration : UserControl
{
public SystemConfiguration()
{
DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<SystemConfigViewModel>();

InitializeComponent();
}
}
}
```

ConnectionDiagram.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
```

```
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for ConnectionDiagram.xaml
/// </summary>
public partial class ConnectionDiagram : UserControl
{
public ConnectionDiagram()
{
InitializeComponent();
}
}
}
```

Digital_Cable_Map.xaml.cs

```
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows;
using System.Windows.Input;
using Telerik.Windows.Diagrams.Core;
using MerlinTestStudio_Demo_Telerik.Resources.DiagramFiles;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Diagrams.Extensions.ViewModels;
using System;
using System.Windows.Threading;

namespace MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager
{
/// <summary>
/// Interaction logic for Digital_Cable_Map.xaml
/// </summary>
public partial class Digital_Cable_Map : UserControl
{
private Telerik.Windows.Controls.Diagrams.Extensions.FileManager fileManager;
private TablesGraphSource dc;
```

```
private bool isClear;

static Digital_Cable_Map()
{
var saveBinding = new CommandBinding(DiagramCommands.Save, ExecuteSave,
CanExecutedSave);
var openBinding = new CommandBinding(DiagramCommands.Open, ExecuteOpen);

CommandManager.RegisterClassCommandBinding(typeof(Digital_Cable_Map), saveBinding);
CommandManager.RegisterClassCommandBinding(typeof(Digital_Cable_Map), openBinding);
}

public Digital_Cable_Map()
{
DiagramConstants.RoutingGridSize = 40d;
DiagramConstants.ContainerMargin = 0d;
InitializeComponent();

this.fileManager = new
Telerik.Windows.Controls.Diagrams.Extensions.FileManager(this.diagram);
diagram.AddShape(new TableShape() { Width = 100, Height = 50, Position = new Point(0, 0) });
diagram.AddShape(new TableShape() { Width =100, Height = 50, Position = new Point(0, 0) });
this.DataContext = this.dc = new TablesGraphSource();
var newRouter = new AStarRouter(this.diagram) { WallOptimization = true };
this.diagram.RoutingService.Router = newRouter;
this.diagram.RoutingService.FreeRouter = newRouter;
this.Loaded += this.OnLoaded;
}

private static void CanExecutedSave(object sender, CanExecuteRoutedEventArgs e)
{
var owner = sender as Digital_Cable_Map;
e.CanExecute = owner != null && owner.diagram != null && owner.diagram.Items.Count > 0;
}

private static void ExecuteSave(object sender, ExecutedRoutedEventArgs e)
{
var owner = sender as Digital_Cable_Map;
if (owner != null)
owner.fileManager.SaveToFile();
}
```

```
private static void ExecuteOpen(object sender, ExecutedRoutedEventArgs e)
{
var owner = sender as Digital_Cable_Map;
if (owner != null)
{
owner.ClearDiagram();
owner.fileManager.LoadFromFile();
}
}

private void OnConnectionManipulationCompleted(object sender, ManipulationRoutedEventArgs
e)
{
if (e.ManipulationPoint.Type != ManipulationPointType.Intermediate)
e.Handled = e.Shape == null;
}

private void OnSelectionChanged(object sender, SelectionChangedEventArgs e)
{
if (e.AddedItems.Count > 0)
{
foreach (var container in this.diagram.Shapes.OfType<RadDiagramContainerShape>())
{
if (container.IsSelected)
container.ZIndex = 4;
else
container.ZIndex = 0;
}
}
}

private void OnPreviewSelectionChanged(object sender, SelectionChangedEventArgs e)
{
if (e.AddedItems.Count > 1)
{
foreach (var item in e.AddedItems)
{
if (item is RowModel) continue;

var container = this.diagram.ContainerGenerator.ContainerFromItem(item);
container.IsSelected = true;
}
```

```
e.Handled = true;
}
}

private void OnNewClick(object sender, RoutedEventArgs e)
{
this.RefreshDiagram();
}

private void OnLoaded(object sender, RoutedEventArgs routedEventArgs)
{
this.RefreshDiagram();
}

private void LoadDefaultTables()
{
//SamplesFactory.LoadSample(this.diagram, "tableShape");
}

private void ClearDiagram()
{
this.diagram.UndoRedoService.Clear();
var graphSource = this.DataContext as TablesGraphSource;
if (graphSource != null)
{
graphSource.ClearCache();
graphSource.Clear();
}
}

private void OnItemsChanging(object sender, DiagramItemsChangingEventArgs e)
{
if (this.isClear) return;

if (e.Action == System.Collections.Specialized.NotifyCollectionChangedAction.Remove)
{
var rowModel = e.OldItems.ElementAt(0) as RowModel;
var command = new CompositeCommand("Remove Connections");
if (rowModel != null)
{
this.RemoveRowModel(rowModel, command);
if (command.Commands.Count() > 0)
```

```
this.diagram.UndoRedoService.ExecuteCommand(command);
}
else
{
var tableModel = e.OldItems.ElementAt(0) as TableModel;
if (tableModel != null)
{
foreach (var item in tableModel.InternalItems)
{
this.RemoveRowModel(item as RowModel, command);
}
if (command.Commands.Count() > 0)
this.diagram.UndoRedoService.ExecuteCommand(command);

tableModel.InternalItems.Clear();
}
}
}
}

private void RemoveRowModel(RowModel rowModel, CompositeCommand command)
{
var container = this.diagram.ContainerGenerator.ContainerFromItem(rowModel) as IShape;
if (container == null) return;

foreach (var connection in this.diagram.GetConnectionsForShape(container).ToList())
{
var link = this.diagram.ContainerGenerator.ItemFromContainer(connection) as
LinkViewModelBase<NodeViewModelBase>;
command.AddCommand(new UndoableDelegateCommand(
"Remove link",
new Action<object>((o) => this.dc.RemoveLink(link)),
new Action<object>((o) => this.dc.AddLink(link))));
}
}

private void RefreshDiagram()
{
this.isClear = true;
this.diagram.DeselectAll();
this.ClearDiagram();
this.LoadDefaultTables();
```

```
this.isClear = false;
}

private void OnDiagramDeserialized(object sender, RadRoutedEventArgs e)
{
// We need to update the connections after the containers have been collapsed.
this.Dispatcher.BeginInvoke(new Action(() =>
{
foreach (var connection in this.diagram.Connections)
{
connection.Update();
}

}), DispatcherPriority.ApplicationIdle);
}
}
}
```

PinMap.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Reflection;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
using Unity;

namespace MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager
{
/// <summary>
/// Interaction logic for PinMap.xaml
/// </summary>
public partial class PinMap : UserControl, IPinMapView
{
private PinMapViewModel viewModelObj;
```

```
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private ObservableCollection<ContextMenuItem> ContextMenuItems = new
ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Insert" },
new ContextMenuItem() { Text = "Delete" }
//new MenuItem() { Text = "Duplicate" }
};

public PinMap(object dataContext)
{
InitializeComponent();

///Set the source then set selected maybe...
PinMapSectionListBox.ItemsSource = new List<string>
{
"RF Pins",
"Digital Pins",
"Power Supply Pins"
};
//PinMapSectionListBox.SelectedItem = PinMapSectionListBox.Items[0];

//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<PinMapViewModel>();
this.DataContext = dataContext;
viewModelObj = dataContext as PinMapViewModel;
viewModelObj.View = this as IPinMapView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.PinMapGrid.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.PinMapGrid);

PinMapGrid.Deleting += PinMapGrid_Deleting;
PinMapGrid.CellEditEnded += PinMapGrid_CellEditEnded;

//Needs IView and viewModelObj property upon reconstruction.
}

#region Context Menu
private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
```

```
{
if (ClickedRow != null && ClickedRow.IsInEditMode == false) //ClickedRow.IsInEditMode == false;
Prevents major bug!
{
foreach (var item in this.ContextMenuItems) { item.IsEnabled = true; }
GridContextMenu.ItemsSource = ContextMenuItems;
if (!PinMapGrid.SelectedItems.Contains(ClickedRow.DataContext as Data.PinModel))
{
PinMapGrid.SelectedCells.Clear();
PinMapGrid.SelectedItem = ClickedRow.DataContext as Data.PinModel;
}
}
else
{
//foreach (var item in this.ContextMenuItems)
//{
//    switch (item.Text)
//    {
//        case "Insert": item.IsEnabled = true; break;
//        case "Delete": item.IsEnabled = false; break;
//        //case "Duplicate": item.IsEnabled = false; break;
//    }
//}
//GridContextMenu.ItemsSource = ContextMenuItems;
GridContextMenu.ItemsSource = null;
}
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
switch (item.Text)
{
case "Insert": viewModelObj.AddPinCommand.Execute(PinMapGrid.SelectedItems); break;
case "Delete": viewModelObj.RemovePinCommand.Execute(PinMapGrid.SelectedItems); break;
//case "Duplicate": viewModelObj.DuplicatePinCommand.Execute(PinMapGrid.SelectedItems);
break;
}
}
#endregion
private void PinMapGrid_Deleting(object sender, GridViewDeletingEventArgs e)
{
```

```
viewModelObj.RemovePinCommand.Execute((sender as RadGridView).SelectedItems);
e.Cancel = true;
}
private void PinMapGrid_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
#region PinMap Multi Edit
try
{
if (e.EditAction == GridViewEditAction.Commit)
{
string FirstPropertyChanged = e.Cell.Column.UniqueName.Split(' ')[0];

//Log the old value(s) and set the new value(s) of each CalPoint Selected.
for (int i = 0; i < PinMapGrid.SelectedCells.Count; i++)
{
GridViewCellInfo cell = PinMapGrid.SelectedCells[i];
GridViewColumnGroup group = PinMapGrid.ColumnGroups.First(g => g.Name ==
cell.Column.ColumnGroupName);
int siteIndex = PinMapGrid.ColumnGroups.IndexOf(group) - 1; // - 1 due to the PIN INFO column
group.

if (siteIndex >= 0) //Is Site Column Group
{
Data.PinModel pin = cell.Item as Data.PinModel;
Data.MappingModel mapping = pin.mappings[siteIndex];

string cellPropertyName = cell.Column.UniqueName.Split(' ')[0];
object OldValue;

//PropertyInfo firstProp = mapping.GetType().GetProperty(FirstPropertyChanged); //Different
objects pin and mapping null comparison for properties.
PropertyInfo cellProp = mapping.GetType().GetProperty(cellPropertyName);

if (FirstPropertyChanged == cellPropertyName)
{
OldValue = cellProp.GetValue(mapping); //Get old value before it's overwrote.
if (i > 0) //Skip the first cell as the grid already commits the data.
{
cellProp.SetValue(mapping, e.NewData);
}

//This fixes a problem with the item ending the edit overwriting the old value with the new one.
```

```
if (pin == e.Cell.ParentRow.Item) { OldValue = e.OldData; }
}
}
else //Is Pin Info Column group, Use as Pin Obj
{
object pinObj = cell.Item;
string cellPropertyName = cell.Column.UniqueName;
object OldValue;

//PropertyInfo firstProp = pinObj.GetType().GetProperty(FirstPropertyChanged); //Different objects pin and mapping null comparison for properties.
PropertyInfo cellProp = pinObj.GetType().GetProperty(cellPropertyName);

if (FirstPropertyChanged == cellPropertyName)
{
OldValue = cellProp.GetValue(pinObj); //Get old value before it's overwrote.
if (i > 0) //Skip the first cell as the grid already commits the data.
{
cellProp.SetValue(pinObj, e.NewData);
}

//This fixes a problem with the item ending the edit overwriting the old value with the new one.
if (pinObj == e.Cell.ParentRow.Item) { OldValue = e.OldData; }
}
}
}

//Forces OnChangesMade when the user changes the edits a CalPoint.
viewModelObj.OnChangeOccured();
}
}
catch(Exception ex)
{
MessageBox.Show(ex.Message);
Console.WriteLine("Error occured executing a pin map multi-edit action.");
}

#endregion
}

public RadGridView GetGridView()
{
```

```
return this.PinMapGrid;
}

public void UnselectAll()
{
this.PinMapGrid.UnselectAll();
}

public void ForceCommitEdit()
{
this.PinMapGrid.CommitEdit();
}

private void PinMapGrid_PreparedCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//This stops the user from selecting rows and accidently making a multi-edit on all the selected
cells within, also while preserving action logging for undo capability.
if (PinMapGrid.SelectedItems.Count() > 0)
{
var cell = PinMapGrid.CurrentCell;
PinMapGrid.SelectedItems.Clear();
PinMapGrid.SelectedCells.Add(new GridViewCellInfo(cell));
}
}



//IN PROGRESS
private void PinMapGrid_CellValidating(object sender, GridViewCellValidatingEventArgs e)
{
try
{
//double newDataParsed = double.Parse(e.NewValue.ToString());
//dynamic DynExObj = e.Row.Item;
//double Baseline = (double)DynExObj.Baseline;
//
//switch (e.Cell.Column.UniqueName)
//{
//    case "Min":
//        if (newDataParsed >= Baseline)
//        {
//            e.IsValid = false;
```

```
//            e.ErrorMessage = "Min cannot be greater than or equal to the Baseline.";
//        }
//        else { e.IsValid = true; }
//        break;
//    case "Max":
//        if (newDataParsed <= Baseline)
//        {
//            e.IsValid = false;
//            e.ErrorMessage = "Max cannot be less than or equal to the Baseline.";
//        }
//        else { e.IsValid = true; }
//        break;
//    case "MinTolerance":
//    case "MaxTolerance":
//        if (newDataParsed < 0) //Check for negative value.
//        {
//            e.IsValid = false;
//            e.ErrorMessage = "Tolerance cannot be set as a negative value.";
//        }
//        else { e.IsValid = true; }
//        break;
//}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
}
}
```

RF_Cable_Map.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.GraphViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.Diagrams;
using Telerik.Windows.Controls.Diagrams.Extensions.ViewModels;
using Telerik.Windows.Diagrams.Core;

namespace MerlinTestStudio_Demo_Telerik.UserControls.ConnectionManager
```

```
{
/// <summary>
/// Interaction logic for RF_Cable_Map.xaml
/// </summary>
public partial class RF_Cable_Map : UserControl, IDiagramView
{
private DiagramViewModel viewModelObj;
//DiagramViewModel DVM = new DiagramViewModel();
//List<RadDiagramConnector> connectorList = new List<RadDiagramConnector>();

public RF_Cable_Map(DiagramViewModel dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = dataContext;

((DiagramViewModel)this.DataContext).View = this;

if(diagram.Items.Count == 0)
{
this.diagram.GraphSource = new GraphSource();
}

//tree.ItemsSource = diagram.GraphSource.Items;

this.diagram.Loaded += ContainerGenerator_StatusChanged;
}

//Saves the diagram to a string so we can write it to an XML file.
public string GetDiagramSerString()
{
return this.diagram.Save();
}

//Loads the diagram by setting a Deserialization string.
public void SetDiagramDeSerString(string DeSerString)
{
this.diagram.Load(DeSerString);
}
```

```
//public GraphSource GetGraphSource()
//{
//    return this.diagram.GraphSource as GraphSource;
//}
//
//public void SetGraphSource(GraphSource graphSourceToSet)
//{
//    if (graphSourceToSet != null)
//    {
//        diagram.GraphSource = graphSourceToSet;
//    }
//    else { Console.WriteLine("Cannot set GraphSource to null"); }
//
//}

private void ContainerGenerator_StatusChanged(object sender, EventArgs e)
{
if (this.diagram.ContainerGenerator.Status == GeneratorStatus.ContainersGenerated)
{
foreach (object node in diagram.GraphSource.Items)
{
if (node is Port)
{
var shape = diagram.ContainerGenerator.ContainerFromItem(node) as RadDiagramShape;
if (shape != null)
{
shape.Connectors.Clear();
var connectorCollection = diagram.Resources["SecondCollection"] as
CustomConnectorCollection;
if (connectorCollection != null)
{
foreach (RadDiagramConnector connector in connectorCollection)
{
var newConnector = new RadDiagramConnector() { Offset = connector.Offset, Name =
connector.Name + "_" + shape.GetHashCode().ToString() };
shape.Connectors.Add(newConnector);
}
}
}
}
}
}
```

```
}

private void diagram_ConnectorActivationChanged(object sender,
ConnectorActivationChangedEventArgs e)
{
//viewModelObj.OnChangeOccured();
}

private void diagram_ItemsChanged(object sender, DiagramItemsChangedEventArgs e)
{
//viewModelObj.OnChangeOccured();
}

private void diagram_PositionChanged(object sender, PositionChangedRoutedEventArgs e)
{
//viewModelObj.OnChangeOccured();
}

//private void InitializeConnections()
//{
//    int Name = 0; // temp int for naming
//    Diagram.Shapes.ToList().ForEach(x =>
//    {
//        //x.Connectors.Clear();
//
//        int n = 0;
//        double d = 0;
//        var i = 0.1;
//        while (n != 10 )
//        {
//            RadDiagramConnector a = new RadDiagramConnector() { Offset = new Point(1, d),
Name = ("Shape" + Name.ToString() + "Connector" + n.ToString())};
//
//            connectorList.Add(a);
//            d = (d + i);
//            ++n;
//        }
//
//        foreach (RadDiagramConnector RDC in connectorList)
//        {
//            x.Connectors.Add(RDC);
//        }
```

```
//
//        ++Name;
//    });
//}
}
}
```

AboutBox.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Diagnostics;
using System.Windows.Navigation;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for AboutBox.xaml
/// </summary>
public partial class AboutBox : Window
{
/// <summary>
/// Constructor for the About Box Window.
/// </summary>
public AboutBox()
{
InitializeComponent();

// Set the XAML's data context to be this class.
DataContext = this;
}
```

```
/// <summary>
/// Gets the current version of the GUI assembly.
/// </summary>
public string Version
{
get
{
return Assembly.GetExecutingAssembly().GetName().Version.ToString();
}
}

/// <summary>
/// Method that's invoked when the 'OK ' button is clicked and closes the about box.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void okButton_Click(object sender, RoutedEventArgs e)
{
this.Close();
}

/// <summary>
/// Method that's invoked when the link in the about box is clicked which open's a browser to the
clicked link.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Hyperlink_RequestNavigate(object sender, RequestNavigateEventArgs e)
{
Process.Start(new ProcessStartInfo(e.Uri.AbsoluteUri));
e.Handled = true;
}
}
}
```

AddNewFileDialog.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels.DialogViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Dialogs
{
/// <summary>
/// Interaction logic for AddNewFileDialog.xaml
/// </summary>
public partial class AddNewFileDialog : UserControl
{
public AddNewFileDialog(AddNewFileDialogVM viewModel)
{
DataContext = viewModel;

InitializeComponent();
}

private void RadButton_Click(object sender, RoutedEventArgs e)
{
CloseWindow();
}

private void NewItemListBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
if(this.NewItemListBox.SelectedItem != null)
{
(this.DataContext as AddNewFileDialogVM).AddCommand.Execute(null);
CloseWindow();
}
}

private void CloseWindow()
{
```

```
RadWindow window = this.ParentOfType<RadWindow>();
window.Close();
}

}
}
```

CloseProjectSaveDiaglog.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for CloseProjectSaveDiaglog.xaml
/// </summary>
public partial class CloseProjectSaveDiaglog : UserControl
{
public CloseProjectSaveDiaglog()
{
InitializeComponent();
}
}
}
```

ImportPinsWindowContent.xaml.cs

```
using System.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
```

```
/// <summary>
/// Interaction logic for ImportPinsWindowContent.xaml
/// </summary>
public partial class ImportPinsWindowContent : UserControl
{
public ImportPinsWindowContent()
{
InitializeComponent();
}
}
}
```

NewProjectDiaglog.xaml.cs

```
using MT.TestStudio.GUI.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Unity;

namespace MT.TestStudio.GUI.User_Controls
{
/// <summary>
/// Interaction logic for NewProjectDiaglog.xaml
/// </summary>
public partial class NewProjectDiaglog : UserControl
{
public NewProjectDiaglog()
{
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<NewProjectDiaglogViewModel>()
;
```

```
InitializeComponent();
}

private void ProjectNameTextBox_GotMouseCapture(object sender, MouseEventArgs e)
{
// Fixes issue when clicking cut/copy/paste in context menu
if (ProjectNameTextBox.SelectionLength == 0)
ProjectNameTextBox.SelectAll();
}

private void ProjectNameTextBox_LostMouseCapture(object sender, MouseEventArgs e)
{
// If user highlights some text, don't override it
if (ProjectNameTextBox.SelectionLength == 0)
ProjectNameTextBox.SelectAll();

// further clicks will not select all
ProjectNameTextBox.LostMouseCapture -= ProjectNameTextBox_LostMouseCapture;
}

private void ProjectNameTextBox_LostKeyboardFocus(object sender,
KeyboardFocusChangedEventArgs e)
{
// once we've left the TextBox, return the select all behavior
ProjectNameTextBox.LostMouseCapture += ProjectNameTextBox_LostMouseCapture;
}
}
}
```

NewProjectDialog.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Dialogs
{
/// <summary>
/// Interaction logic for NewProjectDialog.xaml
/// </summary>
public partial class NewProjectDialog : UserControl
{
public NewProjectDialog()
{
InitializeComponent();
}
}
}
```

RemoveSiteWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls
{
/// <summary>
/// Interaction logic for RemoveSiteWindow.xaml
/// </summary>
public partial class RemoveSiteWindow : UserControl
{
```

```
public RemoveSiteWindow()
{
InitializeComponent();
}

private void RadButton_Click(object sender, RoutedEventArgs e)
{
RadWindow window = this.ParentOfType<RadWindow>();
window.Close();
}
}
}
```

CalResultsExplorer.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for CalResultsExplorer.xaml
/// </summary>
public partial class CalResultsExplorer : UserControl
{
private MainWindowViewModel MVM { get { return (DataContext as PaneViewModel).MVM; } }
```

```
private List<ContextMenuItem> NullContextMenuSource => new List<ContextMenuItem>() { new
ContextMenuItem() { Text = "Import Results", IsEnabled = true }, };
private List<ContextMenuItem> CalDataModelContextMenuSource = new
List<ContextMenuItem>()
{
new ContextMenuItem() {  Text = "Create Limits File", IsEnabled = false },
new ContextMenuItem() { Text = "Remove", IsEnabled = true }
};

RadTreeViewItem lastRightClickedItem = null;
#region Constructor
public CalResultsExplorer()
{
InitializeComponent();

EventManager.RegisterClassHandler(typeof(RadTreeViewItem),
FrameworkElement.MouseRightButtonDownEvent, new
MouseButtonEventHandler(MainWindow_MouseRightButtonDown));
}
#endregion

void MainWindow_MouseRightButtonDown(object sender, MouseButtonEventArgs e)
{
if (this.lastRightClickedItem != null)
{
this.lastRightClickedItem.IsSelected = false;
}

RadTreeViewItem treeViewItem = sender as RadTreeViewItem;
treeViewItem.IsSelected = true;
treeViewItem.Focus();
this.lastRightClickedItem = treeViewItem;

e.Handled = true;
}

private void RadTreeView_ItemDoubleClick(object sender, Telerik.Windows.RadRoutedEventArgs
e)
{
//Do Nothing...
}
```

```
//Chooses which context menu items to use based on what the item is, Context menu options should be sourced by the item being clicked on in the future.
private void TreeViewContextMenu_Opened(object sender, RoutedEventArgs e)
{
if (ResultsTree.SelectedItem != null)
{
if (ResultsTree.SelectedItem is CalResult)
{
TreeViewContextMenu.ItemsSource = null;
}
else if (ResultsTree.SelectedItem is CalDataModel)
{
#region Check if CalDataModel results contain any of the whitelisted names
foreach (var result in ((CalDataModel)ResultsTree.SelectedItem).Results)
{
if (BackendConstants.CalLimit_Producible_Results.Contains(result.ResultName))
{
CalDataModelContextMenuSource[0].IsEnabled = true; //Switch "Create Limits File" menu item IsEnabled to true.
break;
}
else { CalDataModelContextMenuSource[0].IsEnabled = false; }
}
#endregion Check if CalDataModel results contain any of the whitelisted names

TreeViewContextMenu.ItemsSource = CalDataModelContextMenuSource;
}
}
else
{
TreeViewContextMenu.ItemsSource = NullContextMenuSource;
}
}

private void TreeViewContextMenu_ItemClick(object sender,
Telerik.Windows.RadRoutedEventArgs e)
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
switch (item.Text)
{
case "Remove":
MVM.RemoveCalResultCommand.Execute(ResultsTree.SelectedItem);
```

```
break;
case "Import Results":
MVM.ImportResultsCommand.Execute(null);
break;
case "Create Limits File":
MVM.CreateCalLimitsModelCommand.Execute(ResultsTree.SelectedItem); //Send the
corresponding Cal Data Model Results.
break;
}
}
}
}
```

ErrorList.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for ErrorList.xaml
/// </summary>
public partial class ErrorList : UserControl
{
public ErrorList()
{
InitializeComponent();
}
}
}
```

Output.xaml.cs

```
using SFP_UserControlLibrary.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for Output.xaml
/// </summary>
public partial class Output : UserControl
{
ConsoleRedirectWriter consoleRedirectWriter = new ConsoleRedirectWriter();
String LastConsoleString;

public Output()
{
InitializeComponent();
}

private void textBoxDebug_Initialized(object sender, EventArgs e)
{
// Use this for thread safe objects or UIElements in a single thread program
consoleRedirectWriter.OnWrite += delegate (string value) { LastConsoleString = value;
textBoxDebug.AppendText(value); textBoxDebug.ScrollToEnd(); };

/// Multithread operation - Use the dispatcher to write to WPF UIElements if there is more than 1
thread.
///consoleRedirectWriter.OnWrite += Dispatcher.BeginInvoke(DispatcherPriority.Normal,
(Action<string>)delegate (string value) { textBoxDebug.AppendText(value);
```

```
textBoxDebug.ScrollToEnd(); },text );
}


private void RadButton_Click(object sender, RoutedEventArgs e)
{
textBoxDebug.Text = string.Empty;
}
}
}
```

ProjectExplorer.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Models;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.TreeView;
using Telerik.Windows.DragDrop;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for ProjectExplorer.xaml
/// </summary>
public partial class ProjectExplorer : UserControl
{
private MainWindowViewModel MVM { get { return (DataContext as PaneViewModel).MVM; } }

private List<ContextMenuItem> PaneViewContextMenuSource
```

```
{
get
{
return new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = true, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
}
}
private List<ContextMenuItem> FolderContextMenuSource
{
get
{
return new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add New", IsEnabled = true, Command =
MVM.AddNewFileCommand }, //DataStorage.MainViewModel.AddNewFileCommand
new ContextMenuItem() { Text = "Add Existing", IsEnabled = true, Command =
MVM.AddExistingFileCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Import", IsEnabled = true, Command =
MVM.ImportAndExtractDataCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
};
}
}
private List<ContextMenuItem> SolutionContextMenuSource
{
get
{
```

```
return new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Format Test Data", IsEnabled = true, Command =
MVM.FormatProjectTestDataCommand },
//DataStorage.MainViewModel.FormatProjectTestDataCommand
new ContextMenuItem() { Text = "Properties", IsEnabled = true, Command =
MVM.ViewPropertiesCommand }, //DataStorage.MainViewModel.ViewPropertiesCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Close Project", IsEnabled = true, Command =
MVM.UnloadMerlinProjectCommand }
//DataStorage.MainViewModel.UnloadMerlinProjectCommand
};
}
}
private List<ContextMenuItem> NullContextMenuSource
{
get
{
return new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "New Project", IsEnabled = true, Command =
MVM.NewProjectFileCommand }, //DataStorage.MainViewModel.NewProjectFileCommand
new ContextMenuItem() { Text = "Open Existing Project", IsEnabled = true, Command =
MVM.AddExistingMerlinProjectCommand },
//DataStorage.MainViewModel.AddExistingMerlinProjectCommand
};
}
}

RadTreeViewItem lastRightClickedItem = null;
public ProjectExplorer()
{
InitializeComponent();

//Cancel Drop.
DragDropManager.AddDropHandler(this.ProjectTree, OnDrop, true);

//Cancel Drag Start.
DragDropManager.AddDragInitializeHandler(this.ProjectTree, OnTreeViewDragInitialize, true);

//Only disallows drop inside for PVMs, allows drop inside for folders
DragDropManager.AddDragOverHandler(ProjectTree, OnDragOver, true); //Enable only drop
```

```
inside for folders.

//Selects items upon right click.
EventManager.RegisterClassHandler(typeof(RadTreeViewItem),
FrameworkElement.MouseRightButtonDownEvent, new
MouseButtonEventHandler(MainWindow_MouseRightButtonDown));
}

#region TreeView Drag Drop Events
private void OnDrop(object sender, Telerik.Windows.DragDrop.DragEventArgs e)
{
TreeViewDragDropOptions options = DragDropPayloadManager.GetDataFromObject(e.Data,
TreeViewDragDropOptions.Key) as TreeViewDragDropOptions;
if (options != null && options.DragSourceItem != null && options.DropTargetItem != null &&
options.DropPosition != DropPosition.Undefined)
{
RadTreeViewItem dropSourceItem = options.DragSourceItem; //Item being dragged.
RadTreeViewItem dropTargetItem = options.DropTargetItem; //Target item to drop in.

if(dropSourceItem.DataContext is PVM) //Should only be PVM.
{
PVM sourcePVM = (PVM)dropSourceItem.DataContext;

if (dropTargetItem.DataContext is ProjectFolder && options.DropPosition == DropPosition.Inside )
{
ProjectFolder targetFolder = (ProjectFolder)dropTargetItem.DataContext;
//If folder section matches, drop within parent folder, other wise cancel drop. (only exeption being
waveforms)
switch (sourcePVM.ParentFolder.Section)
{
case Data.ProjectFileSection.Waveforms:
case Data.ProjectFileSection.WaveformSubFolder:
options.DropAction = targetFolder.Section == Data.ProjectFileSection.Waveforms ||
targetFolder.Section == Data.ProjectFileSection.WaveformSubFolder ? DropAction.Move :
DropAction.None;
break;
default:
options.DropAction = sourcePVM.ParentFolder.Section == targetFolder.Section ?
DropAction.Move : DropAction.None;
break;
}
}
```

```
else if (dropTargetItem.DataContext is PVM && dropTargetItem != dropSourceItem)
{
ProjectFolder targetFolder = ((PVM)dropTargetItem.DataContext).ParentFolder;
//If folder section matches, drop within parent folder, other wise cancel drop. (only exeption being
waveforms)
switch (sourcePVM.ParentFolder.Section)
{
case Data.ProjectFileSection.Waveforms:
case Data.ProjectFileSection.WaveformSubFolder:
options.DropAction = targetFolder.Section == Data.ProjectFileSection.Waveforms ||
targetFolder.Section == Data.ProjectFileSection.WaveformSubFolder ? DropAction.Move :
DropAction.None;
break;
default:
options.DropAction = sourcePVM.ParentFolder.Section == targetFolder.Section ?
DropAction.Move : DropAction.None;
break;
}
}
else
{
options.DropAction = DropAction.None;
}
}
else
{
options.DropAction = DropAction.None;
}
}
}

private void OnTreeViewDragInitialize(object sender, DragInitializeEventArgs e)
{
TreeViewDragDropOptions options = DragDropPayloadManager.GetDataFromObject(e.Data,
TreeViewDragDropOptions.Key) as TreeViewDragDropOptions;
if (options != null && options.DragSourceItem != null)
{
RadTreeViewItem draggedItem = options.DragSourceItem;
if (draggedItem.DataContext is ProjectFolder || draggedItem.DataContext is MerlinProject)
//Cancel drag start since Project and Folder are not modifiable.
{
e.Data = null;
```

```
e.DragVisual = null;
}
}
}

private void OnDragOver(object sender, Telerik.Windows.DragDrop.DragEventArgs e)
{
var options = DragDropPayloadManager.GetDataFromObject(e.Data,
TreeViewDragDropOptions.Key) as TreeViewDragDropOptions;
if (options != null && options.DropTargetItem.DataContext is PVM && options.DropPosition ==
Telerik.Windows.Controls.DropPosition.Inside)
{
options.DropPosition = DropPosition.Undefined; //places nowhere
options.UpdateDragVisual();
}
}

#endregion TreeView Drag Drop Events

void MainWindow_MouseRightButtonDown(object sender, MouseButtonEventArgs e)
{
if (this.lastRightClickedItem != null) //Left click not stored.
{
this.lastRightClickedItem.IsSelected = false;
}

RadTreeViewItem treeViewItem = sender as RadTreeViewItem;
treeViewItem.IsSelected = true;
treeViewItem.Focus();
this.lastRightClickedItem = treeViewItem;

e.Handled = true;
}

//Empty.
private void RadTreeView_ItemDoubleClick(object sender, Telerik.Windows.RadRoutedEventArgs
e)
{
//Do Nothing...
}

//Chooses which context menu items to use based on what the item is, Context menu options
```

```
should be sourced by the item being clicked on in the future.
private void RadContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ProjectTree.SelectedItems.Last() is null)
{
TreeViewContextMenu.ItemsSource = NullContextMenuSource;
return;
}

//Switch to using the .last() of the selected items, instead of selected item, to fix wrong context
menu on multi selection bug.
object lastSelectedItem = ProjectTree.SelectedItems.Last();

if (lastSelectedItem is PVM)
{
var pfs = (lastSelectedItem as PVM).ParentFolder.Section;
switch(pfs)
{
case Data.ProjectFileSection.Waveforms:
case Data.ProjectFileSection.WaveformSubFolder:
var wfMVM = (lastSelectedItem as PaneViewModel).ViewDataContext as
WaveformConverterControls.MainWindowViewModel;
if (System.IO.Path.GetExtension((lastSelectedItem as PaneViewModel).DataFilePath) == ".aiq")
{
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand },
new ContextMenuItem() { Text = "Save as .mtwf2", IsEnabled = true, Command =
MVM.SaveAsMTWFCommand},
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false }
};
}
else
{
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
```

```
{
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand },
new ContextMenuItem() { Text = "Save as .aiq", IsEnabled = true, Command =
MVM.SaveAsAIQCommand},
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false }
};
}
break;
case Data.ProjectFileSection.Product_Configurations:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = true, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Calibration_Definitions:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = true, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = true, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
```

```
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Connection_Manager:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = true, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = false, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Test_Parameters:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = false, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = false, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
```

```
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Test_Sequences:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = false, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = false, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Calibration_Limits:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Duplicate", IsEnabled = false, Command =
MVM.DuplicateProjectFileItemCommand },
new ContextMenuItem() { Text = "Rename", IsEnabled = false, Command =
MVM.RenameFileCommand }, //DataStorage.MainViewModel.RenameFileCommand
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy", IsEnabled = false },
new ContextMenuItem() { Text = "Paste", IsEnabled = false },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
```

```
break;
case Data.ProjectFileSection.Dll:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>() //TEMP
{
new ContextMenuItem() { Text = "Delete", IsEnabled = true, Command =
MVM.DeleteFileCommand }, //DataStorage.MainViewModel.DeleteFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Copy Full Path", IsEnabled = true, Command =
MVM.CopyFullPathCommand },
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Properties", IsEnabled = false },
};
break;
case Data.ProjectFileSection.Test_Limits:
case Data.ProjectFileSection.Digital_Patterns:
case Data.ProjectFileSection.Connection_Diagrams:
TreeViewContextMenu.ItemsSource = PaneViewContextMenuSource;
break;
default:
TreeViewContextMenu.ItemsSource = null;
break;
}
}
else if (lastSelectedItem is ProjectFolder)
{
var pfs = (lastSelectedItem as ProjectFolder).Section;
switch (pfs)
{
case Data.ProjectFileSection.None:
case Data.ProjectFileSection.WaveformSubFolder:
case Data.ProjectFileSection.Connection_Manager:
case Data.ProjectFileSection.Test_Parameters:
case Data.ProjectFileSection.Test_Sequences:
TreeViewContextMenu.ItemsSource = null;
break;
case Data.ProjectFileSection.Product_Configurations:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add New", IsEnabled = true, Command =
MVM.AddNewFileCommand }, //DataStorage.MainViewModel.AddNewFileCommand
new ContextMenuItem() { Text = "Add Existing", IsEnabled = true, Command =
MVM.AddExistingFileCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
```

```
};
break;
case Data.ProjectFileSection.Dll:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Reference", IsEnabled = true, Command =
MVM.AddExistingFileCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
};
break;
case Data.ProjectFileSection.Waveforms:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Waveform", IsEnabled = true, Command =
MVM.AddExistingFileCommand }
};
break;
case Data.ProjectFileSection.Calibration_Limits:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add New", IsEnabled = false, Command =
MVM.AddNewFileCommand }, //DataStorage.MainViewModel.AddNewFileCommand
new ContextMenuItem() { Text = "Add Existing", IsEnabled = true, Command =
MVM.AddExistingFileCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
new ContextMenuItem() { IsSeparator = true, IsEnabled = true },
new ContextMenuItem() { Text = "Import", IsEnabled = false, Command =
MVM.ImportAndExtractDataCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
};
break;
case Data.ProjectFileSection.Connection_Diagrams:
TreeViewContextMenu.ItemsSource = new List<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add New", IsEnabled = true, Command =
MVM.AddNewFileCommand }, //DataStorage.MainViewModel.AddNewFileCommand
new ContextMenuItem() { Text = "Add Existing", IsEnabled = true, Command =
MVM.AddExistingFileCommand }, //DataStorage.MainViewModel.AddExistingFileCommand
};
break;
default:
TreeViewContextMenu.ItemsSource = FolderContextMenuSource;
break;
}
}
```

```
else if (lastSelectedItem is MerlinProject)
{
TreeViewContextMenu.ItemsSource = SolutionContextMenuSource;
}
}
catch (Exception ex)
{
Console.WriteLine(ex.Message);
TreeViewContextMenu.ItemsSource = NullContextMenuSource;
}
}

private void RadContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
//Could pass in the Command Parameter here if dynamic, just a matter of what fires first, menu item command or this event.
switch (item.Text)
{
//case "Delete":
case "Add Reference":
case "Import":
case "Copy Full Path":
case "Add New":
case "Add Existing":
case "Add Waveform":
case "Rename":
case "Format Test Data":
case "Properties":
case "Close Project":
case "Save as .aiq":
case "Save as .mtwf2":
case "Duplicate":
item.Command.Execute(ProjectTree.SelectedItems.Last());
break;
case "Delete":
item.Command.Execute(ProjectTree.SelectedItems);
break;
case "New Project":
case "Open Existing Project":
item.Command.Execute(null);
break;
```

```
default:
//Execute no command to avoid errors.
break;
}
}
}
}
```

PropertiesPane.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for PropertiesPane.xaml
/// </summary>
public partial class PropertiesPane : UserControl
{
public PropertiesPane()
{
InitializeComponent();
}
}
}
```

SystemExplorer.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.PaneViews
{
/// <summary>
/// Interaction logic for SystemExplorer.xaml
/// </summary>
public partial class SystemExplorer : UserControl
{
private MainWindowViewModel MVM { get { return (DataContext as PaneViewModel).MVM; } }
public SystemExplorer()
{
InitializeComponent();
}
}
}
```

DigitalLevel.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
```

```
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Patterns
{
/// <summary>
/// Interaction logic for DigitalLevels.xaml
/// </summary>
public partial class DigitalLevel : UserControl, IDigitalLevelsView
{
private DigitalLevelViewModel viewModelObj;
public DigitalLevel(DigitalLevelViewModel dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = DataContext as DigitalLevelViewModel;

viewModelObj.View = this;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.DigitalLevelsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.DigitalLevelsGridView);
this.PPMULevelsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.PPMULevelsGridView);
this.DCPowerLevelsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.PPMULevelsGridView);
}

//Gets fired by three different grids.
private void GridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;

if((gridView.ItemsSource as IEnumerable<object>).Count() == 0) //If Empty Add a row to each grid
if grid is empty.
```

```
{
if (gridView.ItemsSource is ObservableCollection<DigiLevel>)
{
(gridView.ItemsSource as ObservableCollection<DigiLevel>).Add(new DigiLevel());
}
else if (gridView.ItemsSource is ObservableCollection<PPMULevel>)
{
(gridView.ItemsSource as ObservableCollection<PPMULevel>).Add(new PPMULevel());
}
else if (gridView.ItemsSource is ObservableCollection<DCPowerLevel>)
{
(gridView.ItemsSource as ObservableCollection<DCPowerLevel>).Add(new DCPowerLevel());
}
}
}

private void GridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
RadGridView gridView = (RadGridView)sender;

//This Adds a new Item To the given grid view if the user edits the last row.
if (gridView.Items.IndexOf(e.Row.Item) == (gridView.Items.ItemCount - 1)) //Check to see if the
last item in which ever grid is being edited;
{
if(gridView.ItemsSource is ObservableCollection<DigiLevel>)
{
(gridView.ItemsSource as ObservableCollection<DigiLevel>).Add(new DigiLevel());
}
else if (gridView.ItemsSource is ObservableCollection<PPMULevel>)
{
(gridView.ItemsSource as ObservableCollection<PPMULevel>).Add(new PPMULevel());
}
else if (gridView.ItemsSource is ObservableCollection<DCPowerLevel>)
{
(gridView.ItemsSource as ObservableCollection<DCPowerLevel>).Add(new DCPowerLevel());
}
}
}

private void GridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
```

```
viewModelObj.CurrentLevelsData.OnChangeOccured(); //Needed since the data lists are just strings.
}

public void ForceCommitEdit()
{
this.DigitalLevelsGridView.CommitEdit();
this.PPMULevelsGridView.CommitEdit();
this.DCPowerLevelsGridView.CommitEdit();
}
}
}
```

DigitalPattern.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
using Telerik.Windows.SyntaxEditor.Core.Text;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Patterns
{
/// <summary>
/// Interaction logic for DigitalPattern.xaml
/// </summary>
```

```
public partial class DigitalPattern : UserControl, IDigitalPatternView
{
private ObservableCollection<ContextMenuItem> ColumnContextMenuItems;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;
private DigitalPatternVM viewModelObj;

public DigitalPattern(DigitalPatternVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = DataContext as DigitalPatternVM;

viewModelObj.View = this as IDigitalPatternView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.DigitalPatternsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.DigitalPatternsGridView);

ColumnContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Register" },
new ContextMenuItem() { Text = "Rename Register" },
new ContextMenuItem() { Text = "Delete Register" }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Mode"},
new ContextMenuItem() {Text = "Remove Mode"},
new ContextMenuItem() {Text = "Duplicate Mode"},
};

}

private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }
```

```
private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
if (ClickedColumn != null) // Open parameter context menu items.
{
string header = (ClickedColumn.Content as TextBlock).Text;

foreach (string regHeader in viewModelObj.CurrentPattern.Registers)
{
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedColumn != null)
{
string header = (ClickedColumn.Content as TextBlock).Text;
int headerIndex = -1;

foreach (string regHeader in viewModelObj.CurrentPattern.Registers) //Get index of
if (header == regHeader)
headerIndex = viewModelObj.CurrentPattern.Registers.IndexOf(regHeader);

if (headerIndex != -1)
{
var regHeader = viewModelObj.CurrentPattern.Registers[headerIndex];
switch (item.Text)
{
case "Delete Register":
GridViewDataColumn gvdc = null;
foreach (var column in from GridViewDataColumn column in DigitalPatternsGridView.Columns
where column.Header.ToString() == header
select column)
{
gvdc = column;
}
```

```
if (gvdc != null)
{
DigitalPatternsGridView.Columns.Remove(gvdc);
viewModelObj.RemoveRegisterCommand.Execute(regHeader);
}
break;
case "Rename Register":
viewModelObj.RenameRegisterCommand.Execute(regHeader);
break;
case "Add Register":
viewModelObj.AddRegisterCommand.Execute(regHeader);
break;
default:
break;
}
}
}
else if (ClickedRow != null)
{
Mode mode = ClickedRow.DataContext as Mode;
switch (item.Text)
{
case "Add Mode":
viewModelObj.AddModeCommand.Execute(mode);
break;
case "Remove Mode":
viewModelObj.RemoveModeCommand.Execute(mode);
break;
case "Duplicate Mode":
//viewModelObj.DuplicateTestCommand.Execute(mode);
break;
}
}
else
{

}
}

private void GridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
```

```
gridView.CurrentItem = null;
}

private void GridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//RadGridView gridView = (RadGridView)sender;

//This Adds a new Item To the given grid view if the user edits the last row.
//if (gridView.Items.IndexOf(e.Row.Item) == (gridView.Items.ItemCount - 1)) //Check to see if the
last item in which ever grid is being edited;
//{
//    if (gridView.ItemsSource is ObservableCollection<Mode>)
//    {
//        (gridView.ItemsSource as ObservableCollection<Mode>).Add(new Mode());
//    }
//}
}

//Temporary Solution to getting time set name data from project tree.
private void SelectTimeSetBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
viewModelObj.ForceGetTimeSets();
}

private void DigitalPatternsGridView_CellEditEnded(object sender,
GridViewCellEditEndedEventArgs e)
{
if (e.EditAction == GridViewEditAction.Commit)
{
GridViewRowItem row = e.Cell.ParentRow;
if (row != null && row.Item is Mode)
{
Mode item = row.Item as Mode;

//Use column index to find register index and vector index since we have the parentmode.
ModeVector vector = null;
int index = DigitalPatternsGridView.Columns.IndexOf(e.Cell.Column) - GUI_Column_Count();
if(index >= 0)
{
vector = item.ModeVectors[index];
if (vector != null)
```

```
{
vector.InputValue = e.NewData as string;
viewModelObj.CurrentPattern.Vectorize_Single(vector, index);
}
}

item.FindDetectedOpcodes();
viewModelObj.CurrentPattern.Compile_Single_Mode_(item); //Rebuilt on row cell edit ended.
viewModelObj.SelectedItem = item;
}
}

viewModelObj.CurrentPattern.OnChangeOccured(); //Needed since the data lists are just strings.
}

public void AddColumn(string columnName)
{
string headerText = columnName;

int i = DigitalPatternsGridView.Columns.Count - 1; //-1 Accounts for the name column.
DigitalPatternsGridView.Columns.Add(new GridViewDataColumn()
{
Header = headerText,
DataMemberBinding = new Binding("ModeVectors[" + i + "].InputValue")
{
Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
}
});
}

public void InsertColumn(int index, string columnName)
{
string headerText = columnName;
int GUI_Index = index + 1;
DigitalPatternsGridView.Columns.Insert(GUI_Index, new GridViewDataColumn()
{
Header = headerText,
DataMemberBinding = new Binding("ModeVectors[" + index + "].InputValue")
{
Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
```

```
}
});
}

public void RemoveAtColumn(int index)
{
int GUI_Index = index + 1;
DigitalPatternsGridView.Columns.RemoveAt(GUI_Index);
}

//helper to calculate GUI column problems with static-dynamic mixtures. USE LATER.
private int GUI_Column_Count()
{
return DigitalPatternsGridView.Columns.Count - viewModelObj.CurrentPattern.Registers.Count();
}

/// <summary>
/// Clears the columns in the GridView.
/// </summary>
public void ResetTable()
{
DigitalPatternsGridView.Columns.Clear();
}

public void RebindGridView()
{
DigitalPatternsGridView.Rebind();
}

public void ForceCommitEdit()
{
this.DigitalPatternsGridView.CommitEdit();
}

//TEMP
public void ShowDPatSrcInEditor(string dpatsrcdata)
{
this.syntaxEditor.Document = new TextDocument(dpatsrcdata);
}



}
```

}

DigitalTiming.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Patterns
{
/// <summary>
/// Interaction logic for DigitalTiming.xaml
/// </summary>
public partial class DigitalTiming : UserControl, IDigitalTimingView
{
private DigitalTimingVM viewModelObj;

public DigitalTiming(DigitalTimingVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = DataContext as DigitalTimingVM;

viewModelObj.View = this;
```

```
this.DigiTimingGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.DigiTimingGridView);
}

private void GridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void GridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//RadGridView gridView = (RadGridView)sender;
//
////This Adds a new Item To the given grid view if the user edits the last row.
//if (gridView.Items.IndexOf(e.Row.Item) == (gridView.Items.ItemCount - 1)) //Check to see if the
last item in which ever grid is being edited;
//{
//    if (gridView.ItemsSource is ObservableCollection<DigiTimeObj>)
//    {
//        (gridView.ItemsSource as ObservableCollection<DigiTimeObj>).Add(new DigiTimeObj());
//    }
//}
}

private void DigiTimingGridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs
e)
{
try
{
if (e.EditAction == GridViewEditAction.Commit)
{
if (e.Cell.Column.UniqueName == "Period")
{
GridViewCellInfo cell = this.DigiTimingGridView.SelectedCells[0];
string timeSetName = (cell.Item as DigiTimeObj).Name;
double newPeriod = (double)e.NewData;

//Get list of all digitimeobjs with the same name
List<DigiTimeObj> itemsToSync = new List<DigiTimeObj>();
```

```
foreach (DigiTimeObj dto in viewModelObj.CurrentTimingData.DigiTimeSource) //LINQ extract desired.
{
if(dto.Name == timeSetName)
{
itemsToSync.Add(dto);
}
}

//Go through the newly created list and change all the "Periods" to be the same.
foreach(var dtoSync in itemsToSync)
{
dtoSync.Period = (double)e.NewData;
}
}

//viewModelObj.DataObject.OnChangeOccured();
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); Console.WriteLine(ex.Message); }

viewModelObj.CurrentTimingData.OnChangeOccured(); //Needed since the data lists are just strings.
}

public void ForceCommitEdit()
{
this.DigiTimingGridView.CommitEdit();
}
}
}
```

GenericRffePattern.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.Data.Models.PatternModels;
using MerlinTestStudio_Demo_Telerik.ViewModels.PatternViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
using Telerik.Windows.Data;
using Telerik.Windows.SyntaxEditor.Core.Text;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Patterns
{
/// <summary>
/// Interaction logic for DigitalPattern.xaml
/// </summary>
public partial class GenericRffePattern : UserControl, IGenericRffePatternView
{
private ObservableCollection<ContextMenuItem> ModeContextMenuItems;
private ObservableCollection<ContextMenuItem> ModeVectorContextMenuItems;
private GenericRffePatternVM viewModelObj;

public GenericRffePattern(GenericRffePatternVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = DataContext as GenericRffePatternVM;

viewModelObj.View = this as IGenericRffePatternView;

GridViewTableDefinition td = new GridViewTableDefinition() { Relation = new
PropertyRelation("ModeVectors") };
DigitalPatternsGridView.ChildTableDefinitions.Add(td);
//RowReorderBehavior.SetIsEnabled(DigitalPatternsGridView, true);

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
```

```
//this.DigitalPatternsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.DigitalPatternsGridView);

ModeContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Insert Mode", IsEnabled = true },
new ContextMenuItem() {Text = "Remove Mode", IsEnabled = true },
new ContextMenuItem() {Text = "Duplicate Mode", IsEnabled = false },
new ContextMenuItem() {IsSeparator = true },
new ContextMenuItem() {Text = "Add Data", IsEnabled = true },
};

//Icon paths not working.
ModeVectorContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
//new ContextMenuItem() { IsSeparator = true },
new ContextMenuItem() {Text = "Insert Data", IsEnabled = true },
new ContextMenuItem() {Text = "Remove Data", IsEnabled = true },
};

}

//private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
//if (ClickedColumn != null) // Open parameter context menu items.
//{
//    string header = (ClickedColumn.Content as TextBlock).Text;
//
//    foreach (string regHeader in viewModelObj.CurrentPattern.Registers)
//    {
//            GridContextMenu.ItemsSource = ColumnContextMenuItems;
//    }
//}
if (ClickedRow != null) //Open test context menu items.
{
if (ClickedRow.DataContext is GenericMode)
{
```

```
GridContextMenu.ItemsSource = ModeContextMenuItems;
}
else if (ClickedRow.DataContext is GenericModeVector)
{
GridContextMenu.ItemsSource = ModeVectorContextMenuItems;
}
}
else //No context menu items.
{
GridContextMenu.ItemsSource = null;
}
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedRow != null)
{
if (ClickedRow.DataContext is GenericMode)
{
GenericMode mode = ClickedRow.DataContext as GenericMode;
switch (item.Text)
{
case "Insert Mode":
viewModelObj.InsertModeCommand.Execute(DigitalPatternsGridView.SelectedItems);
break;
case "Remove Mode":
viewModelObj.RemoveModeCommand.Execute(mode);
break;
case "Duplicate Mode":
//viewModelObj.DuplicateTestCommand.Execute(mode);
break;
case "Add Data":
viewModelObj.AddDataCommand.Execute(mode);
break;
}
}
else if (ClickedRow.DataContext is GenericModeVector)
{
GenericModeVector modeVector = ClickedRow.DataContext as GenericModeVector;
switch (item.Text)
{
```

```
case "Insert Data":
viewModelObj.InsertDataCommand.Execute(modeVector);
break;
case "Remove Data":
viewModelObj.RemoveDataCommand.Execute(modeVector);
break;
}
}
}
}

private void GridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

//Currently Not Used.
private void GridView_PreparingCellForEdit(object sender,
GridViewPreparingCellForEditEventArgs e)
{
//RadGridView gridView = (RadGridView)sender;

//This Adds a new Item To the given grid view if the user edits the last row.
//if (gridView.Items.IndexOf(e.Row.Item) == (gridView.Items.ItemCount - 1)) //Check to see if the
last item in which ever grid is being edited;
//{
//    if (gridView.ItemsSource is ObservableCollection<Mode>)
//    {
//        (gridView.ItemsSource as ObservableCollection<Mode>).Add(new Mode());
//    }
//}
}

//Temporary Solution to getting time set name data from project tree.
private void SelectTimeSetBox_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
viewModelObj.ForceGetTimeSets();
}

//Master Digital Pattern Grid View
private void DigitalPatternsGridView_CellEditEnded(object sender,
```

```
GridViewCellEditEndedEventArgs e)
{
if (e.EditAction == GridViewEditAction.Commit)
{
GridViewRowItem row = e.Cell.ParentRow;
if (row != null && row.Item is GenericMode)
{
GenericMode item = row.Item as GenericMode;

////Use column index to find register index and vector index since we have the parentmode.
//ModeVector vector = null;
//int index = DigitalPatternsGridView.Columns.IndexOf(e.Cell.Column) - GUI_Column_Count();
//if(index >= 0)
//{
//    vector = item.ModeVectors[index];
//    if (vector != null)
//    {
//        vector.InputValue = e.NewData as string;
//        viewModelObj.CurrentPattern.Vectorize_Single(vector, index, item.Spec);
//    }
//}

//TEMP for demo
//viewModelObj.CurrentPattern.Vectorize_All_For_Mode(item);

item.FindDetectedOpcodes();
viewModelObj.CurrentPattern.Compile_Single_Mode_(item); //Rebuilt on row cell edit ended.
viewModelObj.SelectedItem = item;
}

}

viewModelObj.CurrentPattern.OnChangeOccured(); //Needed since the data lists are just strings.
}

//Child Heirarchy Grid View
private void ChildGridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
if (e.EditAction == GridViewEditAction.Commit)
{
GridViewRowItem row = e.Cell.ParentRow;
//var someParent = row.Parent;
```

```
if (row != null && row.Item is GenericModeVector)
{
GenericModeVector modeVectorRow = row.Item as GenericModeVector;

viewModelObj.CurrentPattern.Vectorize_Single(modeVectorRow);

//TEMP until better way is found, possibly via datacontext of the item row hosting the child grid view.
GenericMode parentMode = null;
foreach (GenericMode mode in viewModelObj.CurrentPattern.ModeCollection)
{
foreach (GenericModeVector mv in mode.ModeVectors)
{
if (mv == modeVectorRow)
{
parentMode = mode;
}
}
}

parentMode.FindDetectedOpcodes();
viewModelObj.CurrentPattern.Compile_Single_Mode_(parentMode); //Rebuilt on row cell edit ended.
viewModelObj.SelectedItem = parentMode;

}
}

viewModelObj.CurrentPattern.OnChangeOccured(); //Needed since the data lists are just strings.
}

//REMOVE.
//public void AddColumn(string columnName)
//{
//    string headerText = columnName;
//
//    int i = DigitalPatternsGridView.Columns.Count - 2; //-2 Accounts for the name and spec column.
//    DigitalPatternsGridView.Columns.Add(new GridViewDataColumn()
//    {
//        Header = headerText,
//        DataMemberBinding = new Binding("ModeVectors[" + i + "].InputValue")
```

```
//        {
//            Mode = BindingMode.TwoWay,
//            UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
//        }
//    });
//}

////REMOVE.
//public void InsertColumn(int index, string columnName)
//{
//    string headerText = columnName;
//    int GUI_Index = index + 2;
//    DigitalPatternsGridView.Columns.Insert(GUI_Index, new GridViewDataColumn()
//    {
//        Header = headerText,
//        DataMemberBinding = new Binding("ModeVectors[" + index + "].InputValue")
//        {
//            Mode = BindingMode.TwoWay,
//            UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
//        }
//    });
//}
//
//public void RemoveAtColumn(int index)
//{
//    int GUI_Index = index + 2;
//    DigitalPatternsGridView.Columns.RemoveAt(GUI_Index);
//}

////helper to calculate GUI column problems with static-dynamic mixtures. USE LATER.
//private int GUI_Column_Count()
//{
//    return DigitalPatternsGridView.Columns.Count -
(viewModelObj.CurrentPattern.Registers.Count() * 2);
//}

/// <summary>
/// Clears the columns in the GridView.
/// </summary>
public void ResetTable()
{
DigitalPatternsGridView.Columns.Clear();
```

```
}

public void RebindGridView()
{
DigitalPatternsGridView.Rebind();
}

public void ForceCommitEdit()
{
this.DigitalPatternsGridView.CommitEdit();
}

//Custom Columns (Old Code)
public void RebindDigitalPatternsGridView()
{
//try
//{
//    Style ColumnStyle = Application.Current.FindResource("TestLimits_ColumnStyle") as Style;
//Check style!!!!!!!!!!!!!!!!!!
//
//    this.DigitalPatternsGridView.ColumnGroups.Clear();
//    this.DigitalPatternsGridView.Columns.Clear();
//
//    #region Test Info Columns
//    this.DigitalPatternsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name =
"ModeInfo", Header = new TextBlock() { Text = "MODE INFO", VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } });
//    //Add the Test Info columns
//    this.DigitalPatternsGridView.Columns.Add(new GridViewDataColumn()
//    {
//        UniqueName = "Name",
//        Header = "Name",
//        DataMemberBinding = new Binding("Name"),
//        ColumnGroupName = "ModeInfo",
//        Style = ColumnStyle
//    });
//    this.DigitalPatternsGridView.Columns.Add(new GridViewComboBoxColumn()
//    {
//        UniqueName = "Spec",
//        Header = "Spec",
//        DataMemberBinding = new Binding("Spec"),
//        DisplayMemberPath = "Value",
```

```
//        SelectedValueMemberPath = "Key",
//        ItemsSource = (this.DataContext as DigitalPatternVM).PatternSpecKvPs,
//        ColumnGroupName = "ModeInfo",
//        EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick
//    });
//    #endregion Test Info Columns
//
//    #region Hard Bins and Multi Bin Columns
//    var hardBinGroup = new GridViewColumnGroup() { Name = "Registers", Header = new
TextBlock() { Text = "REGISTERS", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } };
//    this.DigitalPatternsGridView.ColumnGroups.Add(hardBinGroup);
//
//    #region Generate Multi Pass Columns
//    if ((this.DataContext as DigitalPatternVM).CurrentPattern.ModeCollection.Count != 0)
//    {
//        (this.DataContext as DigitalPatternVM).RegisterHeaders.Clear(); //Reset column group
name tracking
//        for (int i = 0; i < (this.DataContext as DigitalPatternVM).CurrentPattern.Registers.Count; i++)
//        {
//            string regName = (this.DataContext as DigitalPatternVM).CurrentPattern.Registers[i];
//            hardBinGroup.ChildGroups.Add(new GridViewColumnGroup() { Name = regName,
Header = new TextBlock() { Text = regName, VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
//            (this.DataContext as DigitalPatternVM).RegisterHeaders.Add(regName); //Add Column
group name track
//
//            this.DigitalPatternsGridView.Columns.Add(new GridViewDataColumn()
//            {
//                UniqueName = "SA " + (i + 1),
//                Header = "SA",
//                DataMemberBinding = new Binding("ModeVectors[" + i + "].UserIDValue"),
//                ColumnGroupName = regName,
//                Style = ColumnStyle
//            });
//            this.DigitalPatternsGridView.Columns.Add(new GridViewDataColumn()
//            {
//                UniqueName = "Data " + (i + 1),
//                Header = "Data",
//                DataMemberBinding = new Binding("ModeVectors[" + i + "].InputValue"),
//                ColumnGroupName = regName,
//                Style = ColumnStyle
```

```
//            });
//
//        }
//    }
//    #endregion Generate Multi Pass Columns
//
//    #endregion Hard Bins and Multi Bin Columns
//
//}
//catch (Exception ex) { MessageBox.Show(ex.Message); }
}

//TEMP
public void ShowDPatSrcInEditor(string dpatsrcdata)
{
this.syntaxEditor.Document = new TextDocument(dpatsrcdata);
}



}
}
```

ConditionsData.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Data;
using System.Runtime.Remoting.Messaging;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using MerlinTestStudio_Demo_Telerik.Data;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
using Telerik.Windows.DragDrop;
using UnitConversionLib;
using Unity;
using System.Linq;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestConditions
```

```
{
/// <summary>
/// Interaction logic for ConditionsData.xaml
/// </summary>
public partial class ConditionsData : UserControl, IConditionsView
{
private ObservableCollection<MenuItem> UnitContextMenuItems;
private ObservableCollection<MenuItem> TestRowContextMenuItems;
private ConditionsViewModel viewModelObj;

public ConditionsData(object dataContext)
{
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<ConditionsViewModel>();

InitializeComponent();

this.DataContext = dataContext;

viewModelObj = (DataContext as ConditionsViewModel);

viewModelObj.View = this as IConditionsView;

////Auto Generates the columns manually to avoid duplicates.
//foreach (Data.ParameterMapTag pmt in viewModelObj.PVM.ParentProject.ParameterMapTags)
//    AddColumn(pmt.ColumnName, pmt.Unit);

RowReorderBehavior.SetIsEnabled(ConditionsGrid, true);
DragDropManager.AddDragDropCompletedHandler(ConditionsGrid, new
DragDropCompletedEventHandler(OnDragDropCompleted));

UnitContextMenuItems = new ObservableCollection<MenuItem>()
{
new MenuItem()
{
Text = "Change Unit",
SubItems = new ObservableCollection<MenuItem>()
{
new MenuItem() {
Text = "Volts",
SubItems = new ObservableCollection<MenuItem>()
{
```

```
new MenuItem() { Text = "V" },
new MenuItem() { Text = "dV" },
new MenuItem() { Text = "cV" },
new MenuItem() { Text = "mV" },
new MenuItem() { Text = "uV" },
new MenuItem() { Text = "nV" },
new MenuItem() { Text = "pV" }
}
},
new MenuItem()
{
Text = "Amps",
SubItems = new ObservableCollection<MenuItem>()
{
new MenuItem() { Text = "A" },
new MenuItem() { Text = "dA" },
new MenuItem() { Text = "cA" },
new MenuItem() { Text = "mA" },
new MenuItem() { Text = "uA" },
new MenuItem() { Text = "nA" },
new MenuItem() { Text = "pA" }
}
},
new MenuItem()
{
Text = "Frequency",
SubItems = new ObservableCollection<MenuItem>()
{
new MenuItem() { Text = "THz" },
new MenuItem() { Text = "GHz" },
new MenuItem() { Text = "MHz" },
new MenuItem() { Text = "KHz" },
new MenuItem() { Text = "hHz" },
new MenuItem() { Text = "daHz" },
new MenuItem() { Text = "Hz" }
}
},
new MenuItem()
{
Text = "Power",
SubItems = new ObservableCollection<MenuItem>()
{
```

```
new MenuItem() { Text = "dBm" },
new MenuItem() { Text = "dB" }
}
},
new MenuItem() { Text = "Number" },
new MenuItem() { Text = "Percentage" }
}
},
new MenuItem() { Text = "Delete Test Parameter" }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<MenuItem>()
{
new MenuItem() {Text = "Insert New", IconPath = "/Resources/Images/MenuItems/AddIcon.png"},
new MenuItem() {Text = "Remove", IconPath =
"/Resources/Images/MenuItems/RemoveIcon.png"},
new MenuItem() {Text = "Duplicate", IconPath = "/Resources/Images/MenuItems/Clone.png"},
new MenuItem() {Text = "Test Numeration Break", IconPath = ""}
};
}

private void OnDragDropCompleted(object sender, DragDropCompletedEventArgs e)
{
viewModelObj.RenumerateTestsCommand.Execute(null);
}

private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }



private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedColumn != null) // Open parameter context menu items.
{
string header = (ClickedColumn.Content as TextBlock).Text;

foreach (ParameterMapTag pmt in viewModelObj.ParentProject.ParameterMapTags)
```

```
if (header == pmt.ColumnName || header == $"{pmt.ColumnName} - {pmt.Unit}")
GridContextMenu.ItemsSource = !pmt.IsUnitable ? null : UnitContextMenuItems;
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

//Fires when the unit of a parameter has been changed.
private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
MenuItem item = (e.OriginalSource as RadMenuItem).DataContext as MenuItem;
if (ClickedColumn != null)
{
string header = (ClickedColumn.Content as TextBlock).Text;
int headerIndex = -1;

foreach (ParameterMapTag pmt in viewModelObj.ParentProject.ParameterMapTags)
if (header == pmt.ColumnName || header == $"{pmt.ColumnName} - {pmt.Unit}")
headerIndex = pmt.ColumnIndex;

if (headerIndex != -1)
{
var pmt = viewModelObj.ParentProject.ParameterMapTags[headerIndex];
switch (item.Text)
{
case "Delete Test Parameter":
GridViewDataColumn gvdc = null;
foreach (var column in from GridViewDataColumn column in ConditionsGrid.Columns
where column.Header.ToString() == header
select column)
{
gvdc = column;
}
if (gvdc != null)
{
ConditionsGrid.Columns.Remove(gvdc);
```

```
viewModelObj.DeleteTestParameterCommand.Execute(pmt);
}
break;
default:
string oldUnit = pmt.Unit;
string newUnit = item.Text;

pmt.Unit = newUnit;
string newHeader = pmt.ColumnName + " - " + newUnit;
(ClickedColumn.Content as TextBlock).Text = newHeader;

bool IsOfSameUnitType = false;
foreach (string unit in pmt.Units)
if (pmt.Unit == unit)
IsOfSameUnitType = true;

if (IsOfSameUnitType)
{
MessageBoxResult messageBoxResult = System.Windows.MessageBox.Show("Would you like to
convert all values?", "Conversion Confirmation", System.Windows.MessageBoxButton.YesNo);
if (messageBoxResult == MessageBoxResult.Yes)
foreach (ITest t in viewModelObj.ParentProject.Tests)
try
{
string modifiedValue;
Data.Services.UnitService.ValueUnitModifier(t.Parameters[pmt.ColumnIndex].Value.ToString(),
oldUnit, newUnit, out modifiedValue);
t.Parameters[pmt.ColumnIndex].Value = modifiedValue;
}
catch (Exception ex) { }
}
else
{
string unitType;
Data.Services.UnitService.MatchUnit(newUnit, out unitType, out _);
pmt.UnitType = unitType;
}
break;
}
}
}
else if (ClickedRow != null)
```

```
{
ITest testRow = ClickedRow.DataContext as ITest;
switch (item.Text)
{
case "Insert New":
viewModelObj.InsertTestCommand.Execute(testRow);
break;
case "Remove":
viewModelObj.RemoveTestCommand.Execute(testRow);
break;
case "Duplicate":
viewModelObj.DuplicateTestCommand.Execute(testRow);
break;
case "Test Numeration Break":
viewModelObj.DesignateTestNumerationBreakCommand.Execute(testRow);
break;
}
}
else
{

}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

private void LocationInput_KeyUp(object sender, System.Windows.Input.KeyEventArgs e)
{
if (ConditionsGrid.SelectedItem != null)
ConditionsGrid.BringIndexIntoView((ConditionsGrid.ItemsSource as
ObservableCollection<Test>).IndexOf(ConditionsGrid.SelectedItem as Test));
}



/// <summary>
/// Adds the column to the GridView.
/// </summary>
public void AddColumn(string columnName, string unit)
{
try
{
string headerText = string.IsNullOrEmpty(unit) ? columnName : $"{columnName} - {unit}";
```

```
int i = ConditionsGrid.Columns.Count;
ConditionsGrid.Columns.Add(new GridViewDataColumn()
{
Header = headerText,
DataMemberBinding = new Binding("Parameters[" + i + "].Value")
{
Mode = BindingMode.TwoWay,
UpdateSourceTrigger = UpdateSourceTrigger.PropertyChanged
}
});
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

/// <summary>
/// Clears the columns in the GridView.
/// </summary>
public void ResetTable()
{
ConditionsGrid.Columns.Clear();
}

public void RebindGridView()
{
ConditionsGrid.Rebind();
}

private void UserControl_Unloaded(object sender, System.Windows.RoutedEventArgs e)
{
//Method needs to be updated/rebuilt.
//DataManagement.GroupAttributeCheck(); //Makes sure groups still match (either on tab close or project close?).
//viewModelObj = null;
}

public void ForceCommitEdit()
{
this.ConditionsGrid.CommitEdit();
}

private void ConditionsGrid_Loaded(object sender, RoutedEventArgs e)
```

```
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void ConditionsGrid_CellEditEnded(object sender, GridViewCellEditEndedEventArgs e)
{
(this.DataContext as ConditionsViewModel).OnChangeOccured();
}
}

#region ContextMenu MenuItemObjects

public class MenuItem : INotifyPropertyChanged
{
private bool isEnabled = true;
private string text;
private string iconPath;
private ObservableCollection<MenuItem> subItems;

public bool IsEnabled
{
get
{
return this.isEnabled;
}
set
{
if (this.isEnabled != value)
{
this.isEnabled = value;
this.OnNotifyPropertyChanged("IsEnabled");
}
}
}
public string Text
{
get
{
return this.text;
}
set
```

```
{
if (this.text != value)
{
this.text = value;
this.OnNotifyPropertyChanged("Text");
}
}
}
public string IconPath
{
get
{
return this.iconPath;
}
set
{
if (this.iconPath != value)
{
this.iconPath = value;
this.OnNotifyPropertyChanged("IconPath");
}
}
}
public ObservableCollection<MenuItem> SubItems
{
get
{
if (this.subItems == null)
{
this.subItems = new ObservableCollection<MenuItem>();
}
return this.subItems;
}
set
{
if (this.subItems != value)
{
this.subItems = value;
this.OnNotifyPropertyChanged("SubItems");
}
}
}
```

```
#region INPC Members

public event PropertyChangedEventHandler PropertyChanged;
private void OnNotifyPropertyChanged(string ptopertyName)
{
if (this.PropertyChanged != null)
{
this.PropertyChanged(this, new PropertyChangedEventArgs(ptopertyName));
}
}

#endregion
}

#endregion
}
```

GoldLimits.xaml.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestLimits
```

```
{
/// <summary>
/// Interaction logic for GoldLimits.xaml
/// </summary>
public partial class GoldLimits : UserControl, IGoldLimitsView
{
public GoldLimitsVM viewModelObj;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;

public GoldLimits(GoldLimitsVM dataContext)
{
InitializeComponent();

this.DataContext = dataContext;

viewModelObj = dataContext;

viewModelObj.View = this as IGoldLimitsView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.GoldLimitsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.GoldLimitsGridView);

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Test", IsEnabled = false, Command =
DataStorage.MainViewModel.AddTestToActiveProjectCommand },
new ContextMenuItem() {Text = "Remove Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.RemoveTestFromActiveProjectCommand},
new ContextMenuItem() {Text = "Duplicate Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.DuplicateTestToActiveProjectCommand},
};
}

private CommonColumnHeader ClickedColumnGroup { get { return
GridContextMenu.GetClickedElement<CommonColumnHeader>(); } }
private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }
```

```
private void GoldLimitsGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void GoldLimitsGridView_CellEditEnded(object sender,
Telerik.Windows.Controls.GridViewCellEditEndedEventArgs e)
{
viewModelObj.OnChangeOccured();
}

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedRow != null)
{
GoldLimit limit = ClickedRow.DataContext as GoldLimit;
switch (item.Text)
{
case "Add Test":
case "Remove Test":
case "Duplicate Test":
item.Command.Execute(null);
break;
}
}
}
```

```
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

public void ForceCommitEdit()
{
this.GoldLimitsGridView.CommitEdit();
}
}
}
```

OffsetLimits.xaml.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestLimits
{
/// <summary>
/// Interaction logic for OffsetLimits.xaml
/// </summary>
public partial class OffsetLimits : UserControl , IOffsetLimitsView
{
private OffsetLimitsVM viewModelObj;
private ObservableCollection<ContextMenuItem> ColumnContextMenuItems;
```

```
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;

public OffsetLimits(OffsetLimitsVM dataContext)
{
InitializeComponent();

this.DataContext = dataContext;

viewModelObj = dataContext;

viewModelObj.View = this as IOffsetLimitsView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.OffsetLimitsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.OffsetLimitsGridView);

ColumnContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Site", IsEnabled = true, Command =
viewModelObj.AddOffsetSiteCommand },
new ContextMenuItem() { Text = "Delete Site", IsEnabled = true, Command =
viewModelObj.RemoveOffsetSiteCommand }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Test", IsEnabled = false, Command =
DataStorage.MainViewModel.AddTestToActiveProjectCommand },
new ContextMenuItem() {Text = "Remove Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.RemoveTestFromActiveProjectCommand},
new ContextMenuItem() {Text = "Duplicate Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.DuplicateTestToActiveProjectCommand},
};
}

private CommonColumnHeader ClickedColumnGroup { get { return
GridContextMenu.GetClickedElement<CommonColumnHeader>(); } }
private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }
```

```
private void OffsetLimitsGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void OffsetLimitsGridView_CellEditEnded(object sender,
Telerik.Windows.Controls.GridViewCellEditEndedEventArgs e)
{
viewModelObj.OnChangeOccured();
}

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedColumn != null) //if the framework element is not null.
{
string header = (ClickedColumn.Content as TextBlock).Text;
if (header.Contains("Site"))
{
ColumnContextMenuItems[1].IsEnabled = true;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
else
{
ColumnContextMenuItems[1].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
```

```
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedColumn != null)
{
string header = (ClickedColumn.Content as TextBlock).Text;
int headerIndex = -1;
if (header.Contains("Site"))
{
headerIndex = viewModelObj.OffsetSiteHeaders.FindIndex(x => x == header);
}

switch (item.Text)
{
case "Delete Site":
if (headerIndex != -1)
{
item.Command.Execute(headerIndex);
}
break;
case "Add Site":
item.Command.Execute(null);
break;
default:
break;
}
}
else if (ClickedRow != null)
{
OffsetLimit limit = ClickedRow.DataContext as OffsetLimit;
switch (item.Text)
{
case "Add Test":
case "Remove Test":
case "Duplicate Test":
item.Command.Execute(null);
break;
}
}
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}
```

```
public void RebindGridView()
{
this.OffsetLimitsGridView.Rebind();
}

public void ForceCommitEdit()
{
this.OffsetLimitsGridView.CommitEdit();
}

public void RebindOffsetLimitsGridView()
{
try
{
Style ColumnStyle = Application.Current.FindResource("TestLimits_ColumnStyle") as Style;
Style editorStyle = Application.Current.FindResource("DropDownOnCellEdit") as Style;

this.OffsetLimitsGridView.ColumnGroups.Clear();
this.OffsetLimitsGridView.Columns.Clear();

#region Test Info Columns
this.OffsetLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "TestInfo",
Header = new System.Windows.Controls.TextBlock() { Text = "TEST INFO", VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } });
//Add the Test Info columns
this.OffsetLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestNumber",
Header = "Test Number",
DataMemberBinding = new Binding("TestNumber"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.OffsetLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestName",
Header = "Test Name",
TextAlignment = TextAlignment.Left,
DataMemberBinding = new Binding("TestName"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
```

```
this.OffsetLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "Units",
Header = "Units",
DataMemberBinding = new Binding("Units"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
#endregion Test Info Columns

#region Offset Columns
var hardBinGroup = new GridViewColumnGroup() { Name = "Offsets", Header = new
System.Windows.Controls.TextBlock() { Text = "OFFSETS", VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } };
this.OffsetLimitsGridView.ColumnGroups.Add(hardBinGroup);

this.OffsetLimitsGridView.Columns.Add(new GridViewComboBoxColumn()
{
UniqueName = "ApplyOffset",
Header = "Apply",
DataMemberBinding = new Binding("ApplyOffset"),
ItemsSource = Enum.GetValues(typeof(OffsetControl)).Cast<OffsetControl>(),
ColumnGroupName = "Offsets",
Style = ColumnStyle,
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick,
EditorStyle = editorStyle,
MinWidth = 100
}); ;

#region Generate Offset Site Columns
if (viewModelObj.OffsetLimitsObj.Data.Count != 0)
{
viewModelObj.OffsetSiteHeaders.Clear(); //Reset column group name tracking
for (int i = 0; i < viewModelObj.MaxOffsetSitesCount; i++)
{
viewModelObj.OffsetSiteHeaders.Add(("Site " + (i + 1))); //Add Column group name track
this.OffsetLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "OffsetSite " + (i + 1),
Header = ("Site " + (i + 1)),
DataMemberBinding = new Binding("OffsetSites[" + i + "]"),
ColumnGroupName = "Offsets",
```

```
Style = ColumnStyle
});
}
}
#endregion Generate Multi Pass Columns

#endregion
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
}
}
```

TestFixtures.xaml.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestLimits
{
/// <summary>
/// Interaction logic for TestFixtures.xaml
/// </summary>
public partial class TestFixtures : UserControl, ITestFixturesView
```

```
{
private TestFixturesVM viewModelObj;
private ObservableCollection<ContextMenuItem> ColumnContextMenuItems;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;

public TestFixtures(TestFixturesVM dataContext)
{
InitializeComponent();

this.DataContext = dataContext;

viewModelObj = dataContext;

viewModelObj.View = this as ITestFixturesView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.TestFixturesGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.TestFixturesGridView);

ColumnContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Site", IsEnabled = true, Command =
viewModelObj.AddTestFixtureSiteCommand },
new ContextMenuItem() { Text = "Delete Site", IsEnabled = true, Command =
viewModelObj.RemoveTestFixtureSiteCommand }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Test", IsEnabled = false, Command =
DataStorage.MainViewModel.AddTestToActiveProjectCommand },
new ContextMenuItem() {Text = "Remove Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.RemoveTestFromActiveProjectCommand},
new ContextMenuItem() {Text = "Duplicate Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.DuplicateTestToActiveProjectCommand},
};
}

private CommonColumnHeader ClickedColumnGroup { get { return
GridContextMenu.GetClickedElement<CommonColumnHeader>(); } }
private GridViewHeaderCell ClickedColumn { get { return
```

```
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private void TestFixturesGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void TestFixturesGridView_CellEditEnded(object sender,
Telerik.Windows.Controls.GridViewCellEditEndedEventArgs e)
{
viewModelObj.OnChangeOccured();
}

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedColumn != null) //if the framework element is not null.
{
string header = (ClickedColumn.Content as TextBlock).Text;
if (header.Contains("Site"))
{
ColumnContextMenuItems[1].IsEnabled = true;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
else
{
ColumnContextMenuItems[1].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}
```

```
private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedColumn != null)
{
string header = (ClickedColumn.Content as TextBlock).Text;
int headerIndex = -1;
if (header.Contains("Site"))
{
headerIndex = viewModelObj.TestFixtureSiteHeaders.FindIndex(x => x == header);
}

switch (item.Text)
{
case "Delete Site":
if (headerIndex != -1)
{
item.Command.Execute(headerIndex);
}
break;
case "Add Site":
item.Command.Execute(null);
break;
default:
break;
}
}
else if (ClickedRow != null)
{
TraceLossLimit limit = ClickedRow.DataContext as TraceLossLimit;
switch (item.Text)
{
case "Add Test":
case "Remove Test":
case "Duplicate Test":
item.Command.Execute(null);
break;
}
}
}
```

```
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

public void ForceCommitEdit()
{
this.TestFixturesGridView.CommitEdit();
}

public void RebindGridView()
{
this.TestFixturesGridView.Rebind();
}

public void RebindOffsetLimitsGridView()
{
try
{
Style ColumnStyle = Application.Current.FindResource("TestLimits_ColumnStyle") as Style;
Style editorStyle = Application.Current.FindResource("DropDownOnCellEdit") as Style;

this.TestFixturesGridView.ColumnGroups.Clear();
this.TestFixturesGridView.Columns.Clear();

#region Test Info Columns
this.TestFixturesGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "TestInfo",
Header = new TextBlock() { Text = "TEST INFO", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
//Add the Test Info columns
this.TestFixturesGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestNumber",
Header = "Test Number",
DataMemberBinding = new Binding("TestNumber"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TestFixturesGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestName",
Header = "Test Name",
TextAlignment = TextAlignment.Left,
DataMemberBinding = new Binding("TestName"),
```

```
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TestFixturesGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "Units",
Header = "Units",
DataMemberBinding = new Binding("Units"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
#endregion Test Info Columns

#region Offset Columns
var hardBinGroup = new GridViewColumnGroup() { Name = "TestFixtures", Header = new
TextBlock() { Text = "TEST FIXTURES", VerticalAlignment =  VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } };
this.TestFixturesGridView.ColumnGroups.Add(hardBinGroup);

this.TestFixturesGridView.Columns.Add(new GridViewComboBoxColumn()
{
UniqueName = "ApplyTestFixture",
Header = "Apply",
DataMemberBinding = new Binding("ApplyTestFixture"),
ItemsSource = Enum.GetValues(typeof(OffsetControl)).Cast<OffsetControl>(),
ColumnGroupName = "TestFixtures",
Style = ColumnStyle,
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick,
EditorStyle = editorStyle,
MinWidth = 100
}); ;

#region Generate Offset Site Columns
if (viewModelObj.TestFixturesObj.Data.Count != 0)
{
viewModelObj.TestFixtureSiteHeaders.Clear(); //Reset column group name tracking
for (int i = 0; i < viewModelObj.MaxTestFixtureSitesCount; i++)
{
viewModelObj.TestFixtureSiteHeaders.Add(("Site " + (i + 1))); //Add Column group name track
this.TestFixturesGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestFixtureSite " + (i + 1),
```

```
Header = ("Site " + (i + 1)),
DataMemberBinding = new Binding("TestFixtureSites[" + i + "]"),
ColumnGroupName = "TestFixtures",
Style = ColumnStyle
});
}
}
#endregion Generate Multi Pass Columns

#endregion
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}
}
}
```

TestLimits.xaml.cs

```
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestLimits
{
/// <summary>
/// Interaction logic for TestLimits.xaml
```

```
/// </summary>
public partial class TestLimits : UserControl, ITestLimitsView, IDisposable
{
private TestLimitsVM viewModelObj;
private ObservableCollection<ContextMenuItem> ColumnContextMenuItems;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;

public TestLimits(TestLimitsVM dataContext)
{
InitializeComponent();

DataContext = dataContext;

viewModelObj = dataContext;

((TestLimitsVM)this.DataContext).View = this;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.TestLimitsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.TestLimitsGridView);

ColumnContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Multi-Pass", IsEnabled = true, Command =
viewModelObj.AddMultiPassBinCommand },
new ContextMenuItem() { Text = "Add Multi-Fail", IsEnabled = true, Command =
viewModelObj.AddMultiFailBinCommand },
new ContextMenuItem() { Text = "Delete Multi-Pass", IsEnabled = true, Command =
viewModelObj.RemoveMultiPassBinCommand },
new ContextMenuItem() { Text = "Delete Multi-Fail", IsEnabled = true, Command =
viewModelObj.RemoveMultiFailBinCommand }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Test", IsEnabled = false, Command =
DataStorage.MainViewModel.AddTestToActiveProjectCommand },
new ContextMenuItem() {Text = "Remove Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.RemoveTestFromActiveProjectCommand},
new ContextMenuItem() {Text = "Duplicate Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.DuplicateTestToActiveProjectCommand},
```

```
};
}
~TestLimits()
{
//Console.WriteLine("Test Limits Control Finalized..."); //Casuses non owner thread error which then causes memory leak fatal error on exit. //Need custom console messenger.
}

private CommonColumnHeader ClickedColumnGroup { get { return GridContextMenu.GetClickedElement<CommonColumnHeader>(); } }
private GridViewHeaderCell ClickedColumn { get { return GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return GridContextMenu.GetClickedElement<GridViewRow>(); } }


private void TestLimitsGridView_CellEditEnded(object sender, Telerik.Windows.Controls.GridViewCellEditEndedEventArgs e)
{
(this.DataContext as TestLimitsVM).OnChangeOccured(); //May be needed since Multi Bin objetcs don't yet have INPC.
}

//Needs better sensing to know when to display what options.
private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedColumnGroup != null) //if the framework element is not null.
{
string header = (string)ClickedColumnGroup.FindChildByType<TextBlock>().Text;
if (header.Contains("MULTI"))
{
if (header.Contains("PASS"))
{
ColumnContextMenuItems[2].IsEnabled = true;
ColumnContextMenuItems[3].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
else if (header.Contains("FAIL"))
{
ColumnContextMenuItems[2].IsEnabled = false;
```

```
ColumnContextMenuItems[3].IsEnabled = true;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else
{
ColumnContextMenuItems[2].IsEnabled = false;
ColumnContextMenuItems[3].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else if (ClickedColumn != null)
{
//Incase column group headers disappear...
ColumnContextMenuItems[2].IsEnabled = false;
ColumnContextMenuItems[3].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch(Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedColumnGroup != null)
{
string header = (string)ClickedColumnGroup.FindChildByType<TextBlock>().Text;
int headerIndex = -1;
if (header.Contains("PASS"))
{
headerIndex = (this.DataContext as TestLimitsVM).MultiPassHeaders.FindIndex(x => x ==
header);
//foreach (string regHeader in (this.DataContext as TestLimitsVM).MultiPassHeaders) //Get index
of
//    if (header == regHeader)
```

```
//        headerIndex = (this.DataContext as TestLimitsVM).MultiPassHeaders.IndexOf(regHeader);
}
else if (header.Contains("FAIL"))
{
headerIndex = (this.DataContext as TestLimitsVM).MultiFailHeaders.FindIndex(x => x == header);
//foreach (string regHeader in (this.DataContext as TestLimitsVM).MultiFailHeaders) //Get index of
//    if (header == regHeader)
//        headerIndex = (this.DataContext as TestLimitsVM).MultiFailHeaders.IndexOf(regHeader);
}

switch (item.Text)
{
case "Delete Multi-Pass":
case "Delete Multi-Fail":
if (headerIndex != -1)
{
item.Command.Execute(headerIndex);
//(this.DataContext as TestLimitsVM).RemoveMultiPassBinCommand.Execute(headerIndex);
}
break;
case "Add Multi-Pass":
case "Add Multi-Fail":
item.Command.Execute(null);
//(this.DataContext as TestLimitsVM).AddMultiPassBinCommand.Execute(null);
break;
default:
break;
}

if (headerIndex != -1)
{

}
}
else if (ClickedRow != null)
{
UpperLowerLimit limit = ClickedRow.DataContext as UpperLowerLimit;
switch (item.Text)
{
case "Add Test":
case "Remove Test":
case "Duplicate Test":
```

```
item.Command.Execute(null);
break;
}
}
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void TestLimitsGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

public void RebindGridView()
{
this.TestLimitsGridView.Rebind();
}

public void ForceCommitEdit()
{
this.TestLimitsGridView.CommitEdit();
}

public void RebindTestLimitsGridView()
{
try
{
Style ColumnStyle = Application.Current.FindResource("TestLimits_ColumnStyle") as Style;

this.TestLimitsGridView.ColumnGroups.Clear();
this.TestLimitsGridView.Columns.Clear();

#region Test Info Columns
this.TestLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "TestInfo",
Header = new TextBlock() { Text = "TEST INFO", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
//Add the Test Info columns
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestNumber",
Header = "Test Number",
```

```
DataMemberBinding = new Binding("TestNumber"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestName",
Header = "Test Name",
TextAlignment = TextAlignment.Left,
DataMemberBinding = new Binding("TestName"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "Units",
Header = "Units",
DataMemberBinding = new Binding("Units"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
#endregion Test Info Columns

#region FT QA upper lower Columns
//Add the FT-QA upper/lower columns
this.TestLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "FT", Header
= new TextBlock() { Text = "FT", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
this.TestLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "QA", Header
= new TextBlock() { Text = "QA", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "FTLower",
Header = "Lower",
DataMemberBinding = new Binding("FTLower"),
ColumnGroupName = "FT",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "FTUpper",
```

```
Header = "Upper",
DataMemberBinding = new Binding("FTUpper"),
ColumnGroupName = "FT",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "QALower",
Header = "Lower",
DataMemberBinding = new Binding("QALower"),
ColumnGroupName = "QA",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "QAUpper",
Header = "Upper",
DataMemberBinding = new Binding("QAUpper"),
ColumnGroupName = "QA",
Style = ColumnStyle
});
#endregion FT QA upper lower Columns

#region Hard Bins and Multi Bin Columns
var hardBinGroup = new GridViewColumnGroup() { Name = "HardBin", Header = new TextBlock()
{ Text = "HARD BIN", VerticalAlignment = VerticalAlignment.Center, HorizontalAlignment =
HorizontalAlignment.Center } };
hardBinGroup.ChildGroups.Add(new GridViewColumnGroup() { Name = "Fail", Header = new
TextBlock() { Text = "FAIL", VerticalAlignment = VerticalAlignment.Center, HorizontalAlignment =
HorizontalAlignment.Center } });
this.TestLimitsGridView.ColumnGroups.Add(hardBinGroup);

#region Static Hard Bin Columns
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "HardBinNumber",
Header = "Number",
DataMemberBinding = new Binding("HardBinNumber"),
ColumnGroupName = "Fail",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
```

```
{
UniqueName = "HardBinName",
Header = "Hard Bin Name",
DataMemberBinding = new Binding("HardBinName"),
ColumnGroupName = "Fail",
Style = ColumnStyle
});
//this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
//{
//    UniqueName = "HardBinPF",
//    Header = "Flag",
//    DataMemberBinding = new Binding("HardBinPF"),
//    ColumnGroupName = "Fail",
//    Style = ColumnStyle
//});
#endregion Static Hard Bin Columns

#region Generate Multi Pass Columns
if ((this.DataContext as TestLimitsVM).TestLimitsObj.Data.Count != 0)
{
(this.DataContext as TestLimitsVM).MultiPassHeaders.Clear(); //Reset column group name
tracking
for (int i = 0; i < (this.DataContext as TestLimitsVM).MaxMultiPassBinsCount; i++)
{
hardBinGroup.ChildGroups.Add(new GridViewColumnGroup() { Name = ("Multi Pass " + (i + 1)),
Header = new TextBlock() { Text = ("MULTI PASS " + (i + 1)), VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } });
(this.DataContext as TestLimitsVM).MultiPassHeaders.Add(("MULTI PASS " + (i + 1))); //Add
Column group name track
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiPassHardBinNumber " + (i + 1),
Header = "Number",
DataMemberBinding = new Binding("MultiPassBins[" + i + "].HardBinNumber"),
ColumnGroupName = ("Multi Pass " + (i + 1)),
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiPassHardBinName " + (i + 1),
Header = "Name",
DataMemberBinding = new Binding("MultiPassBins[" + i + "].HardBinName"),
```

```
ColumnGroupName = ("Multi Pass " + (i + 1)),
Style = ColumnStyle
});
//this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
//{
//    UniqueName = "MultiPassHardBinPF " + (i + 1),
//    Header = "Flag",
//    DataMemberBinding = new Binding("MultiPassBins[" + i + "].HardBinPF"),
//    ColumnGroupName = ("Multi Pass " + (i + 1)),
//    Style = ColumnStyle
//});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiPassLower " + (i + 1),
Header = "Lower",
DataMemberBinding = new Binding("MultiPassBins[" + i + "].Lower"),
ColumnGroupName = ("Multi Pass " + (i + 1)),
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiPassUpper " + (i + 1),
Header = "Upper",
DataMemberBinding = new Binding("MultiPassBins[" + i + "].Upper"),
ColumnGroupName = ("Multi Pass " + (i + 1)),
Style = ColumnStyle
});
}
}
#endregion Generate Multi Pass Columns

#region Generate Multi Fail Columns
if ((this.DataContext as TestLimitsVM).TestLimitsObj.Data.Count != 0)
{
(this.DataContext as TestLimitsVM).MultiFailHeaders.Clear();
for (int i = 0; i < (this.DataContext as TestLimitsVM).MaxMultiFailBinsCount; i++)
{
hardBinGroup.ChildGroups.Add(new GridViewColumnGroup() { Name = ("Multi Fail " + (i + 1)),
Header = new TextBlock() { Text = ("MULTI FAIL " + (i + 1)), VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center }  });
(this.DataContext as TestLimitsVM).MultiFailHeaders.Add(("MULTI FAIL " + (i + 1)));
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
```

```
{
UniqueName = "MultiFailSoftBinNumber " + (i + 1),
Header = "Number",
DataMemberBinding = new Binding("MultiFailBins[" + i + "].SoftBinNumber"),
ColumnGroupName = ("Multi Fail " + (i + 1)),
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiFailSoftBinName " + (i + 1),
Header = "Name",
DataMemberBinding = new Binding("MultiFailBins[" + i + "].SoftBinName"),
ColumnGroupName = ("Multi Fail " + (i + 1)),
Style = ColumnStyle
});
//this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
//{
//    UniqueName = "MultiFailSoftBinPF " + (i + 1),
//    Header = "Flag",
//    DataMemberBinding = new Binding("MultiFailBins[" + i + "].SoftBinPF"),
//    ColumnGroupName = ("Multi Fail " + (i + 1)),
//    Style = ColumnStyle
//});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiSoftBinLower " + (i + 1),
Header = "Lower",
DataMemberBinding = new Binding("MultiFailBins[" + i + "].Lower"),
ColumnGroupName = ("Multi Fail " + (i + 1)),
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "MultiSoftBinUpper " + (i + 1),
Header = "Upper",
DataMemberBinding = new Binding("MultiFailBins[" + i + "].Upper"),
ColumnGroupName = ("Multi Fail " + (i + 1)),
Style = ColumnStyle
});
}
}
#endregion
```

```
#endregion Hard Bins and Multi Bin Columns

#region Soft Bin Columns
//Add the Soft Bin Columns
this.TestLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name = "SoftBin",
Header = new TextBlock() { Text = "SOFT BIN", VerticalAlignment = VerticalAlignment.Center,
HorizontalAlignment = HorizontalAlignment.Center } });
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "SoftBinNumber",
Header = "Number",
DataMemberBinding = new Binding("SoftBinNumber"),
ColumnGroupName = "SoftBin",
Style = ColumnStyle
});
this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "SoftBinName",
Header = "Soft Bin Name",
DataMemberBinding = new Binding("SoftBinName"),
ColumnGroupName = "SoftBin",
Style = ColumnStyle
});
//this.TestLimitsGridView.Columns.Add(new GridViewDataColumn()
//{
//    UniqueName = "SoftBinPF",
//    Header = "Flag",
//    DataMemberBinding = new Binding("SoftBinPF"),
//    ColumnGroupName = "SoftBin",
//    Style = ColumnStyle
//});
#endregion Soft Bin Columns
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
//this.DataContext = viewModelObj;
//viewModelObj.View = this as ITestLimitsView;
}
```

```
private void UserControl_Unloaded(object sender, RoutedEventArgs e)
{
//this.DataContext = null;
//viewModelObj.View = null;
//GC.Collect();
}

public void Dispose()
{
this.viewModelObj = null;
this.DataContext = null;
this.ColumnContextMenuItems = null;
this.TestRowContextMenuItems = null;
this.Content = null;
}
}
}
```

TracelossLimits.xaml.cs

```
using MerlinTest.Common.Types;
using MerlinTestStudio.DataModels;
using MerlinTestStudio_Demo_Telerik.Data.Helpers;
using MerlinTestStudio_Demo_Telerik.ViewModels.TestLimitsViewModels;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Telerik.Windows.Controls;
using Telerik.Windows.Controls.GridView;
```

```
namespace MerlinTestStudio_Demo_Telerik.UserControls.TestLimits
{
/// <summary>
/// Interaction logic for TracelossLimits.xaml
/// </summary>
public partial class TracelossLimits : UserControl, ITracelossLimitsView
{
private TracelossLimitsVM viewModelObj;
private ObservableCollection<ContextMenuItem> ColumnContextMenuItems;
private ObservableCollection<ContextMenuItem> TestRowContextMenuItems;

public TracelossLimits(TracelossLimitsVM dataContext)
{
InitializeComponent();

this.DataContext = dataContext;

viewModelObj = dataContext;

viewModelObj.View = this as ITracelossLimitsView;

//Changes the sequence of commands fired in the gridview after a certain key is pressed.
this.TracelossLimitsGridView.KeyboardCommandProvider = new
CustomKeyboardCommandProvider(this.TracelossLimitsGridView);

ColumnContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() { Text = "Add Site", IsEnabled = true, Command =
viewModelObj.AddTracelossSiteCommand },
new ContextMenuItem() { Text = "Delete Site", IsEnabled = true, Command =
viewModelObj.RemoveTracelossSiteCommand }
};

//Icon paths not working.
TestRowContextMenuItems = new ObservableCollection<ContextMenuItem>()
{
new ContextMenuItem() {Text = "Add Test", IsEnabled = false, Command =
DataStorage.MainViewModel.AddTestToActiveProjectCommand },
new ContextMenuItem() {Text = "Remove Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.RemoveTestFromActiveProjectCommand},
new ContextMenuItem() {Text = "Duplicate Test", IsEnabled = false,  Command =
DataStorage.MainViewModel.DuplicateTestToActiveProjectCommand},
```

```
};
}

private CommonColumnHeader ClickedColumnGroup { get { return
GridContextMenu.GetClickedElement<CommonColumnHeader>(); } }
private GridViewHeaderCell ClickedColumn { get { return
GridContextMenu.GetClickedElement<GridViewHeaderCell>(); } }
private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedColumn != null) //if the framework element is not null.
{
string header = (ClickedColumn.Content as TextBlock).Text;
if (header.Contains("Site"))
{
ColumnContextMenuItems[1].IsEnabled = true;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
else
{
ColumnContextMenuItems[1].IsEnabled = false;
GridContextMenu.ItemsSource = ColumnContextMenuItems;
}
}
else if (ClickedRow != null) //Open test context menu items.
GridContextMenu.ItemsSource = TestRowContextMenuItems;
else //No context menu items.
GridContextMenu.ItemsSource = null;
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
ContextMenuItem item = (e.OriginalSource as RadMenuItem).DataContext as ContextMenuItem;
if (ClickedColumn != null)
```

```
{
string header = (ClickedColumn.Content as TextBlock).Text;
int headerIndex = -1;
if (header.Contains("Site"))
{
headerIndex = viewModelObj.TracelossSiteHeaders.FindIndex(x => x == header);
}

switch (item.Text)
{
case "Delete Site":
if (headerIndex != -1)
{
item.Command.Execute(headerIndex);
}
break;
case "Add Site":
item.Command.Execute(null);
break;
default:
break;
}
}
else if (ClickedRow != null)
{
TraceLossLimit limit = ClickedRow.DataContext as TraceLossLimit;
switch (item.Text)
{
case "Add Test":
case "Remove Test":
case "Duplicate Test":
item.Command.Execute(null);
break;
}
}
}
catch (Exception ex) { Console.WriteLine(ex.Message); GridContextMenu.ItemsSource = null; }
}

private void TracelossLimitsGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
```

```
gridView.CurrentItem = null;
}

private void TracelossLimitsGridView_CellEditEnded(object sender,
Telerik.Windows.Controls.GridViewCellEditEndedEventArgs e)
{
viewModelObj.OnChangeOccured();
}

public void ForceCommitEdit()
{
this.TracelossLimitsGridView.CommitEdit();
}

public void RebindGridView()
{
this.TracelossLimitsGridView.Rebind();
}

public void RebindOffsetLimitsGridView()
{
try
{
Style ColumnStyle = Application.Current.FindResource("TestLimits_ColumnStyle") as Style;
Style editorStyle = Application.Current.FindResource("DropDownOnCellEdit") as Style;

this.TracelossLimitsGridView.ColumnGroups.Clear();
this.TracelossLimitsGridView.Columns.Clear();

#region Test Info Columns
this.TracelossLimitsGridView.ColumnGroups.Add(new GridViewColumnGroup() { Name =
"TestInfo", Header = new System.Windows.Controls.TextBlock() { Text = "TEST INFO",
VerticalAlignment = VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center }
});
//Add the Test Info columns
this.TracelossLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestNumber",
Header = "Test Number",
DataMemberBinding = new Binding("TestNumber"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
```

```
});
this.TracelossLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TestName",
Header = "Test Name",
TextAlignment = TextAlignment.Left,
DataMemberBinding = new Binding("TestName"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
this.TracelossLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "Units",
Header = "Units",
DataMemberBinding = new Binding("Units"),
ColumnGroupName = "TestInfo",
Style = ColumnStyle
});
#endregion Test Info Columns

#region Offset Columns
var hardBinGroup = new GridViewColumnGroup() { Name = "Traceloss", Header = new
System.Windows.Controls.TextBlock() { Text = "TRACELOSS", VerticalAlignment =
VerticalAlignment.Center, HorizontalAlignment = HorizontalAlignment.Center } };
this.TracelossLimitsGridView.ColumnGroups.Add(hardBinGroup);

this.TracelossLimitsGridView.Columns.Add(new GridViewComboBoxColumn()
{
UniqueName = "ApplyTraceloss",
Header = "Apply",
DataMemberBinding = new Binding("ApplyTraceLoss"),
ItemsSource = Enum.GetValues(typeof(OffsetControl)).Cast<OffsetControl>(),
ColumnGroupName = "Traceloss",
Style = ColumnStyle,
EditTriggers = Telerik.Windows.Controls.GridView.GridViewEditTriggers.CellClick,
EditorStyle = editorStyle,
MinWidth = 100
}); ;

#region Generate Offset Site Columns
if (viewModelObj.TracelossLimitsObj.Data.Count != 0)
{
```

```
viewModelObj.TracelossSiteHeaders.Clear(); //Reset column group name tracking
for (int i = 0; i < viewModelObj.MaxTracelossSitesCount; i++)
{
viewModelObj.TracelossSiteHeaders.Add(("Site " + (i + 1))); //Add Column group name track
this.TracelossLimitsGridView.Columns.Add(new GridViewDataColumn()
{
UniqueName = "TracelossSite " + (i + 1),
Header = ("Site " + (i + 1)),
DataMemberBinding = new Binding("TracelossSites[" + i + "]"),
ColumnGroupName = "Traceloss",
Style = ColumnStyle
});
}
}
#endregion Generate Multi Pass Columns

#endregion
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

}
}
```

DUT_End_Test.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestSequences
{
```

```
/// <summary>
/// Interaction logic for DUT_End_Test.xaml
/// </summary>
public partial class DUT_End_Test : UserControl
{
public DUT_End_Test()
{
InitializeComponent();
DataContext = new ViewModels.DUT_End_TestViewModel();
}
}
}
```

DUT_Start_Test.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestSequences
{
/// <summary>
/// Interaction logic for DUT_Start_Test.xaml
/// </summary>
public partial class DUT_Start_Test : UserControl
{
public DUT_Start_Test()
{
InitializeComponent();
DataContext = new ViewModels.DUT_Start_TestViewModel();
}
}
```

}

DUT_Test.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using Telerik.Windows.Controls;
using Telerik.Windows.Data;
using System.Collections.ObjectModel;
using System.ComponentModel;
using Telerik.Windows.Controls.GridView;
using Unity;
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System.Windows.Input;
using MerlinTestStudio_Demo_Telerik.Data;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestSequences
{
/// <summary>
/// Interaction logic for DUT_Test.xaml
/// </summary>
public partial class DUT_Test : UserControl, IDUTTestView
{
private ObservableCollection<MenuItem> rowContextMenuItems;

private DUT_TestViewModel viewModelObj;


public DUT_Test(DUT_TestViewModel dataContext)
{

InitializeComponent();

DataContext = dataContext;
//((UnityContainer)Application.Current.Resources["IoC"]).Resolve<DUT_TestViewModel>();

viewModelObj = (DataContext as DUT_TestViewModel);

viewModelObj.View = this as IDUTTestView;


GridViewTableDefinition td = new GridViewTableDefinition() { Relation = new
```

```
PropertyRelation("Tests") };
HierarchicalGridView.ChildTableDefinitions.Add(td);
RowReorderBehavior.SetIsEnabled(HierarchicalGridView, true);
HierarchicalGridView.Drop += DropChanges;
FunctionSequenceListDUT.PreviewDrop += DropChanges; //Custom DragDropBehavior
presenting challenges will fix soon.
FunctionsLibraryList.PreviewDrop += DropChanges;

ObservableCollection<MenuItem> groupItems = new ObservableCollection<MenuItem>()
{
new MenuItem() { Text = "Add to Group" },
new MenuItem() { Text = "Group Matching" },
new MenuItem() { Text = "Break Group" },
new MenuItem() { Text = "Skip Remaining" }
};
this.rowContextMenuItems = groupItems;
}

private void DropChanges(object sender, DragEventArgs e)
{
viewModelObj.ForceOnChangesMade();
}

#region Private Methods

/// <summary>
/// This is called when ever a(n) ComboBox selection is changed in the
GridViewComboBoxColumns.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void OnSelectionChanged(object sender, SelectionChangedEventArgs e)
{
if (e.AddedItems.Count > 0)
{
RadComboBox comboBox = e.OriginalSource as RadComboBox;
if (comboBox.SelectedValue != null)
{
string value = comboBox.SelectedValue.ToString();
if(e.RemovedItems.Count > 0)
{
foreach(string s in viewModelObj.CommandStrings)
```

```
{
if (e.RemovedItems[0].ToString() == s)
{
viewModelObj.PreviouslySelectedCommand = s;
}
}
}
viewModelObj.SelectedInComboBoxChangedCommand.Execute(value);
}
}
}

private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
//Handler added once UC_loaded so data added in before doesn't trigger event for each
ComboBox
HierarchicalGridView.AddHandler(RadComboBox.SelectionChangedEvent, new
System.Windows.Controls.SelectionChangedEventHandler(OnSelectionChanged));
}

private GridViewRow ClickedRow { get { return
GridContextMenu.GetClickedElement<GridViewRow>(); } }

#endregion

private void GridContextMenu_Opened(object sender, RoutedEventArgs e)
{
try
{
if (ClickedRow != null)
{
HierarchicalGridView.SelectedItem = ClickedRow.DataContext;
var selected = (ISequenceItem)HierarchicalGridView.SelectedItem;
foreach (var item in rowContextMenuItems)
if (selected == null) return;
if (selected is IGroupable == true)
{
rowContextMenuItems[0].IsEnabled = false;
rowContextMenuItems[1].IsEnabled = false;
rowContextMenuItems[2].IsEnabled = true;
rowContextMenuItems[3].IsEnabled = true;
GridContextMenu.ItemsSource = rowContextMenuItems;
```

```
}
else if (selected is IGroupable == false)
{
rowContextMenuItems[0].IsEnabled = true;
rowContextMenuItems[1].IsEnabled = true;
rowContextMenuItems[2].IsEnabled = false;
rowContextMenuItems[3].IsEnabled = true;
GridContextMenu.ItemsSource = rowContextMenuItems;
}
else if (selected is SequenceBlock == true)
{
rowContextMenuItems[0].IsEnabled = false;
rowContextMenuItems[1].IsEnabled = false;
rowContextMenuItems[2].IsEnabled = false;
rowContextMenuItems[3].IsEnabled = true;
GridContextMenu.ItemsSource = rowContextMenuItems;
}
}
else
{
foreach (var item in this.rowContextMenuItems)
{
item.IsEnabled = false;
}
GridContextMenu.ItemsSource = rowContextMenuItems;
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

private void GridContextMenu_ItemClick(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
try
{
MenuItem item = (e.OriginalSource as RadMenuItem).DataContext as MenuItem;
ISequenceItem SeqItemRow = ClickedRow.DataContext as ISequenceItem;
switch (item.Text)
{
case "Add to Group":
viewModelObj.AddTestToGroupCommand.Execute(SeqItemRow);
break;
case "Group Matching":
```

```
viewModelObj.GroupMatchingCommand.Execute(SeqItemRow);
break;
case "Break Group":
viewModelObj.BreakGroupCommand.Execute(SeqItemRow);
break;
case "Skip Remaining":
viewModelObj.SkipRemainingCommand.Execute(SeqItemRow);
break;
}
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

private void LocationInput_KeyUp(object sender, System.Windows.Input.KeyEventArgs e)
{
if(HierarchicalGridView.SelectedItem != null)
HierarchicalGridView.BringIndexIntoView((HierarchicalGridView.ItemsSource as
ObservableCollection<ISequenceItem>).IndexOf(HierarchicalGridView.SelectedItem as
ISequenceItem));
}

#region IDUTTestView implementation

public void ClearGridViewFocus()
{
try
{
var scope = System.Windows.Input.FocusManager.GetFocusScope(HierarchicalGridView);
System.Windows.Input.FocusManager.SetFocusedElement(scope, null);
System.Windows.Input.Keyboard.ClearFocus();
}
catch (Exception ex) { MessageBox.Show(ex.Message); }
}

public void ScrollObjectIntoView(ISequenceItem obj)
{
this.HierarchicalGridView.ScrollIntoView(obj);
}

public void RebindGridView()
{
HierarchicalGridView.Rebind();
```

```
}

#endregion

private void FunctionSequenceListDUT_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
FunctionsLibraryList.SelectedItem = null;
}

private void ValMapToggleButton_Checked(object sender, RoutedEventArgs e)
{

}

public void ForceCommitEdit()
{
this.HierarchicalGridView.CommitEdit();
}

private void HierarchicalGridView_Loaded(object sender, RoutedEventArgs e)
{
RadGridView gridView = (RadGridView)sender;
gridView.CurrentItem = null;
}

private void HierarchicalGridView_CellEditEnded(object sender, GridViewCellEditEndedEventArgs
e)
{
(this.DataContext as DUT_TestViewModel).OnChangeOccured();
}
}

#region ContextMenu MenuItemObjects

public class MenuItem : INotifyPropertyChanged
{
private bool isEnabled = true;
private string text;
private ObservableCollection<MenuItem> subItems;

public bool IsEnabled
```

```
{
get
{
return this.isEnabled;
}
set
{
if (this.isEnabled != value)
{
this.isEnabled = value;
this.OnNotifyPropertyChanged("IsEnabled");
}
}
}
public string Text
{
get
{
return this.text;
}
set
{
if (this.text != value)
{
this.text = value;
this.OnNotifyPropertyChanged("Text");
}
}
}
public ObservableCollection<MenuItem> SubItems
{
get
{
if (this.subItems == null)
{
this.subItems = new ObservableCollection<MenuItem>();
}
return this.subItems;
}
set
{
if (this.subItems != value)
```

```
{
this.subItems = value;
this.OnNotifyPropertyChanged("SubItems");
}
}
}

#region INPC Members

public event PropertyChangedEventHandler PropertyChanged;
private void OnNotifyPropertyChanged(string ptopertyName)
{
if (this.PropertyChanged != null)
{
this.PropertyChanged(this, new PropertyChangedEventArgs(ptopertyName));
}
}

#endregion
}

#endregion

}
```

SessionLoad.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestSequences
```

```
{
/// <summary>
/// Interaction logic for SessionLoad.xaml
/// </summary>
public partial class Session_Load : UserControl
{
public Session_Load()
{
InitializeComponent();
DataContext = new ViewModels.SessionLoadViewModel();
}
}
}
```

SessionUnload.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.TestSequences
{
/// <summary>
/// Interaction logic for SessionUnload.xaml
/// </summary>
public partial class SessionUnload : UserControl
{
public SessionUnload()
{
InitializeComponent();
DataContext = new ViewModels.SessionUnloadViewModel();
}
```

```
}
}
```

AutoGeneratedChecker.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Tools
{
/// <summary>
/// Interaction logic for AutoGeneratedChecker.xaml
/// </summary>
public partial class AutoGeneratedChecker : UserControl
{
public AutoGeneratedChecker()
{
InitializeComponent();
}
}
}
```

DataFormattingTool.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
```

```
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Unity;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Tools
{
/// <summary>
/// Interaction logic for DataFormattingTool.xaml
/// </summary>
public partial class DataFormattingTool : UserControl
{
public DataFormattingTool(DataFormattingToolViewModel dataContext)
{
//DataContext =
((UnityContainer)Application.Current.Resources["IoC"]).Resolve<DataFormattingToolViewModel>(
);
InitializeComponent();

this.DataContext = dataContext;
}
}
}
```

FunctionSequencingTool.xaml.cs

```
using MerlinTestStudio_Demo_Telerik.Data.Services;
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Linq;
using Telerik.Windows.Controls;
using MerlinTestStudio_Demo_Telerik.Data.Models;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Tools
{
/// <summary>
/// Interaction logic for FunctionSequencingTool.xaml
```

```
/// </summary>
public partial class FunctionSequencingTool : UserControl
{
MerlinProject PassedProject { get; set; }

public FunctionSequencingTool(MerlinProject project)
{
InitializeComponent();
PassedProject = project;
SourceComboBox.ItemsSource = project.DLLs;
}

public SequenceCollection<Data.Function> PassedSequenceCollection
{
get { return
(SequenceCollection<Data.Function>)this.GetValue(PassedSequenceCollectionProperty); }
set { this.SetValue(PassedSequenceCollectionProperty, value); }
}

public static readonly DependencyProperty PassedSequenceCollectionProperty =
DependencyProperty.Register(
"PassedSequenceCollection", typeof(SequenceCollection<Data.ISequenceItemComponent>),
typeof(FunctionSequencingTool), new PropertyMetadata(new
SequenceCollection<Data.ISequenceItemComponent>()));

private void ImportBtn_Click(object sender, Telerik.Windows.RadRoutedEventArgs e)
{
// Create OpenFileDialog
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog() { Title = "Import
Extension", DefaultExt = ".dll", Filter = "Application Extension (.dll)|*.dll",  };

bool? result = dlg.ShowDialog();

if (result == true)
{
var dll_Types = MerlinProject.ImportAssmblyTypes(dlg.FileName);
foreach (Data.DLL dll in dll_Types)
PassedProject.DLLs.Add(dll);

if (dll_Types.Count > 0)
SourceComboBox.SelectedItem = dll_Types[0]; //Selects the first item of a newly imported group
of dll types.
```

```
}
}

private void CancelBtn_Click(object sender, RoutedEventArgs e)
{
PassedSequenceCollection.Clear(); //Empty the collection.

RadWindow window = this.ParentOfType<RadWindow>();
window.Close();
}
}
}
```

GeneratePinsFromCalibrationDefinition.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MerlinTestStudio_Demo_Telerik.UserControls.Tools
{
/// <summary>
/// Interaction logic for GeneratePinsFromCalibrationDefinition.xaml
/// </summary>
public partial class GeneratePinsFromCalibrationDefinition : UserControl
{
public GeneratePinsFromCalibrationDefinition()
{
InitializeComponent();
}
}
}
```

WaveformConverterControl.xaml.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Forms.DataVisualization.Charting;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WaveformConverterControls
{
/// <summary>
/// Interaction logic for WaveformConverterControl.xaml
/// </summary>
public partial class WaveformConverterControl : UserControl, IView
{
#region Private Member Variables

/// <summary>
/// Instance of this VIEWs VIEW-MODEL;
/// </summary>
private MainWindowViewModel viewModelObj;

#endregion Private Member Variables

//public string FilePathToLoad
//{
//    get { return (string)GetValue(Property1Property); }
//    set { SetValue(Property1Property, value); }
//}
//
```

```
//// Using a DependencyProperty as the backing store for Property1.
//// This enables animation, styling, binding, etc...
//public static readonly DependencyProperty Property1Property
//    = DependencyProperty.Register(
//          "Property1",
//          typeof(string),
//          typeof(WaveformConverterControl),
//          new PropertyMetadata(false)
//      );

public WaveformConverterControl(MainWindowViewModel dataContext)
{
InitializeComponent();

this.DataContext = dataContext;

//  We have declared the view model instance declaratively in the xaml.
//  Get the reference to it here.
viewModelObj = dataContext;

// Subscribe to the VIEW's loaded event and execute a method that in turn will execute a method in the View-Model when the View is loaded.
//this.Loaded += new RoutedEventHandler(MainWindow_Loaded);

// Set the ViewModels view property to be equal to this. This allows the viewModel access to the view but only
// through a controled interface thus not breaking the MVVM pattern.
viewModelObj.View = this as IView;
}

public WaveformConverterControl()
{
InitializeComponent();

this.DataContext = new MainWindowViewModel();

//  We have declared the view model instance declaratively in the xaml.
//  Get the reference to it here.
viewModelObj = (DataContext as MainWindowViewModel);

// Subscribe to the VIEW's loaded event and execute a method that in turn will execute a method in the View-Model when the View is loaded.
```

```
//this.Loaded += new RoutedEventHandler(MainWindow_Loaded);

// Set the ViewModels view property to be equal to this. This allows the viewModel access to the view but only
// through a controled interface thus not breaking the MVVM pattern.
viewModelObj.View = this as IView;
}

public void SetupChart(string Title, string XaxisTitle, string Xaxis2Title, string YaxisTitle, double XaxisMinimum, double Xaxis2Minimum, double XaxisMaximum, double Xaxis2Maximum)
{
panel1Chart.SetupChart(Title, XaxisTitle, Xaxis2Title, YaxisTitle, XaxisMinimum, Xaxis2Minimum, XaxisMaximum, Xaxis2Maximum);
}

public void DrawSeries(double[] xValues, double[] results, int seriesIndex, AxisType axisType)
{
panel1Chart.DrawSeries(xValues, results, seriesIndex, axisType);
}

private void UserControl_Loaded(object sender, RoutedEventArgs e)
{

}
}

}
```

XvsYGenericChart.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Reflection;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Forms.DataVisualization.Charting;
using System.Windows.Media;
using System.Windows.Media.Imaging;

namespace WaveformConverterControls
{
```

```
public class Stat
{
public double Max1 { get; set; }
public double Max2 { get; set; }
public double Min1 { get; set; }
public double Min2 { get; set; }
public double Average1 { get; set; }
public double Average2 { get; set; }
public double StdDev1 { get; set; }
public double StdDev2 { get; set; }
public string Units { get; set; }
}
/// <summary>
/// Class that contains extension methods to the MSChart Control.
/// </summary>
internal static class MSChartExtension
{
/// <summary>
/// Extension method that clears all the points in a series. ( the official method is very slow. )
/// </summary>
/// <param name="sender">The series to clear points </param>
public static void ClearPoints(this System.Windows.Forms.DataVisualization.Charting.Series
sender)
{
// https://www.codeproject.com/Articles/376060/Speedup-MSChart-Clear-Data-Points
sender.Points.SuspendUpdates();
while (sender.Points.Count > 0)
sender.Points.RemoveAt(sender.Points.Count - 1);
sender.Points.ResumeUpdates();
sender.Points.Clear(); //NOTE 1
}
}
/// <summary>
/// Interaction logic for PowerVsTime.xaml
/// </summary>
public partial class XvsYGenericChart : UserControl
{
#region Private Member Variables
private const string DEFAULT_CHART_AREA = "DefaultChartArea";
private const string Marker_CHART_AREA = "MarkerChartArea";
private const string DEFAULT_LEGEND = "DefaultLegend";
private const string DEFAULT_TITLE = "DefaultTitle";
```

```
private const string IMAGE_NAME_CHECKED = "ifchecked";
private const string IMAGE_NAME_UNCHECKED = "ifunchecked";
#endregion Private Member Variables

#region Constructor

/// <summary>
/// Constructor for the XvsYGeneric class.
/// </summary>
public XvsYGenericChart()
{
InitializeComponent();

Chart chartObj = this.FindName("XvsYChart") as Chart;

Assembly assembly = Assembly.GetExecutingAssembly();
chartObj.Images.Add(new NamedImage(IMAGE_NAME_CHECKED,
System.Drawing.Image.FromStream(assembly.GetManifestResourceStream("WaveformConverter
Controls.if_checked_checkbox.png"))));
chartObj.Images.Add(new NamedImage(IMAGE_NAME_UNCHECKED,
System.Drawing.Image.FromStream(assembly.GetManifestResourceStream("WaveformConverter
Controls.if_unchecked_checkbox.png"))));

//default chart area
var defaultArea = chartObj.ChartAreas.Add(DEFAULT_CHART_AREA);
{
defaultArea.CursorX.Interval = 0;
defaultArea.CursorX.IsUserEnabled = true;
defaultArea.CursorX.IsUserSelectionEnabled = true;
defaultArea.CursorX.LineColor = System.Drawing.Color.White;
defaultArea.CursorX.SelectionColor = System.Drawing.Color.Gray;

defaultArea.AxisX2.TitleFont = new Font("Tahoma", 10, System.Drawing.FontStyle.Bold);
defaultArea.AxisX2.TitleForeColor = System.Drawing.Color.WhiteSmoke;
defaultArea.AxisX2.LabelStyle.ForeColor = System.Drawing.Color.Gray;
defaultArea.AxisX2.LabelStyle.Format = "{0.###}";
defaultArea.AxisX2.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisX2.MinorGrid.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisX2.MajorGrid.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisX2.MajorTickMark.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisX2.IsStartedFromZero = true;
defaultArea.AxisX2.IsMarginVisible = false;
```

```
defaultArea.AxisY.TitleFont = new Font("Tahoma", 10, System.Drawing.FontStyle.Bold);
defaultArea.AxisY.TitleForeColor = System.Drawing.Color.WhiteSmoke;
defaultArea.AxisY.LabelStyle.ForeColor = System.Drawing.Color.Gray;
defaultArea.AxisY.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisY.MinorGrid.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisY.MajorGrid.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisY.MajorTickMark.LineColor = System.Drawing.Color.Gray;
defaultArea.AxisY.IsStartedFromZero = false;

defaultArea.BackColor = chartObj.BackColor;
defaultArea.Visible = false;
}

var markerArea = chartObj.ChartAreas.Add(Marker_CHART_AREA);
{
markerArea.CursorX.Interval = 0;
markerArea.CursorX.IsUserEnabled = true;
markerArea.CursorX.IsUserSelectionEnabled = true;
markerArea.CursorX.LineColor = System.Drawing.Color.White;
markerArea.CursorX.SelectionColor = System.Drawing.Color.Gray;

markerArea.AxisX.TitleFont = new Font("Tahoma", 10, System.Drawing.FontStyle.Bold);
markerArea.AxisX.TitleForeColor = System.Drawing.Color.WhiteSmoke;
markerArea.AxisX.LabelStyle.ForeColor = System.Drawing.Color.Gray;
markerArea.AxisX.LineColor = System.Drawing.Color.Gray;
markerArea.AxisX.MinorGrid.LineColor = System.Drawing.Color.Gray;
markerArea.AxisX.MajorGrid.LineColor = System.Drawing.Color.Gray;
markerArea.AxisX.MajorTickMark.LineColor = System.Drawing.Color.Gray;
markerArea.AxisX.IsStartedFromZero = true;
markerArea.AxisX.IsMarginVisible = false;

markerArea.AxisY.TitleFont = new Font("Tahoma", 10, System.Drawing.FontStyle.Bold);
markerArea.AxisY.TitleForeColor = System.Drawing.Color.WhiteSmoke;
markerArea.AxisY.LabelStyle.ForeColor = System.Drawing.Color.Gray;
markerArea.AxisY.LineColor = System.Drawing.Color.Gray;
markerArea.AxisY.MinorGrid.LineColor = System.Drawing.Color.Gray;
markerArea.AxisY.MajorGrid.LineColor = System.Drawing.Color.Gray;
markerArea.AxisY.MajorTickMark.LineColor = System.Drawing.Color.Gray;
markerArea.AxisY.IsStartedFromZero = false;

//This will align inner plot area of marker graph to power graph
```

```
markerArea.AlignWithChartArea = defaultArea.Name;
markerArea.AlignmentStyle = AreaAlignmentStyles.PlotPosition;
markerArea.AlignmentOrientation = AreaAlignmentOrientations.Vertical;

markerArea.BackColor = chartObj.BackColor;
markerArea.Visible = false;
}

// default legend
var defaultLegend = chartObj.Legends.Add(DEFAULT_LEGEND);
{
defaultLegend.BackColor = System.Drawing.Color.Gray;
defaultLegend.BorderColor = System.Drawing.Color.Black;
defaultLegend.BorderWidth = 2;
defaultLegend.BorderDashStyle = ChartDashStyle.Solid;
defaultLegend.ShadowOffset = 2;
defaultLegend.Alignment = StringAlignment.Center;
defaultLegend.LegendStyle = LegendStyle.Table;
defaultLegend.Docking = Docking.Bottom;
//defaultLegend.DockedToChartArea = DEFAULT_CHART_AREA;
defaultLegend.IsDockedInsideChartArea = false;
}

// title
var title = chartObj.Titles.Add(string.Empty);
{
title.Name = DEFAULT_TITLE;
title.Font = new Font("Tahoma", 14, System.Drawing.FontStyle.Bold);
title.ForeColor = System.Drawing.Color.WhiteSmoke;
title.Docking = Docking.Top;
title.DockedToChartArea = DEFAULT_CHART_AREA;
title.IsDockedInsideChartArea = false;
}

// series
chartObj.Series.Add("Power").Color = System.Drawing.Color.Red;
chartObj.Series.Add("Marker 1").Color = System.Drawing.Color.Green;
chartObj.Series.Add("Marker 2").Color = System.Drawing.Color.Blue;
chartObj.Series.Add("Marker 3").Color = System.Drawing.Color.Yellow;
chartObj.Series.Add("Marker 4").Color = System.Drawing.Color.Cyan;

foreach (var series in chartObj.Series)
```

```
{
series.IsVisibleInLegend = false;
series.IsValueShownAsLabel = false;
if (series.Name == "Power")
{
series.ChartArea = DEFAULT_CHART_AREA;
}
else
{
series.ChartArea = Marker_CHART_AREA;
}
series.ChartType = SeriesChartType.FastLine;
series.SmartLabelStyle.Enabled = false;
series.Enabled = false;

LegendItem legendItem = new LegendItem
{
ImageStyle = LegendImageStyle.Rectangle,
Color = series.Color,
BorderColor = series.Color
};
legendItem.Cells.Add(LegendCellType.Image, IMAGE_NAME_UNCHECKED,
ContentAlignment.MiddleCenter);
legendItem.Cells.Add(LegendCellType.SeriesSymbol, series.Name,
ContentAlignment.MiddleCenter);
legendItem.Cells.Add(LegendCellType.Text, series.Name, ContentAlignment.MiddleLeft);

legendItem.Tag = series;

series.Tag = legendItem;

defaultLegend.CustomItems.Add(legendItem);
}

}

#endregion Constructor

#region Public Properties

/// <summary>
/// Gets / Sets the title of the chart.
```

```
/// </summary>
public string Title
{
get
{
Chart chart = this.FindName("XvsYChart") as Chart;
return chart.Titles[DEFAULT_TITLE].Text;
}
set
{
Chart chart = this.FindName("XvsYChart") as Chart;
chart.Titles[DEFAULT_TITLE].Text = value;
}
}

/// <summary>
/// Gets / Sets the X axis title of the chart.
/// </summary>
public string TitleXAxis
{
get
{
Chart chart = this.FindName("XvsYChart") as Chart;
return chart.ChartAreas[DEFAULT_CHART_AREA].AxisX.Title;
}
set
{
Chart chart = this.FindName("XvsYChart") as Chart;
chart.ChartAreas[DEFAULT_CHART_AREA].AxisX.Title = value;
chart.ChartAreas[Marker_CHART_AREA].AxisX.Title = value;
}
}

/// <summary>
/// Gets / Sets the secondary X axis title of the chart.
/// </summary>
public string TitleXAxis2
{
get
{
Chart chart = this.FindName("XvsYChart") as Chart;
return chart.ChartAreas[DEFAULT_CHART_AREA].AxisX2.Title;
```

```
}
set
{
Chart chart = this.FindName("XvsYChart") as Chart;
chart.ChartAreas[DEFAULT_CHART_AREA].AxisX2.Title = value;
chart.ChartAreas[Marker_CHART_AREA].AxisX2.Title = value;
}
}

/// <summary>
/// Gets / Sets the Y axis title of the chart.
/// </summary>
public string TitleYAxis
{
get
{
Chart chart = this.FindName("XvsYChart") as Chart;
return chart.ChartAreas[DEFAULT_CHART_AREA].AxisY.Title;
}
set
{
Chart chart = this.FindName("XvsYChart") as Chart;
chart.ChartAreas[DEFAULT_CHART_AREA].AxisY.Title = value;
chart.ChartAreas[Marker_CHART_AREA].AxisY.Title = "Marker On/Off";
}
}

#endregion Public Properties

#region Public Methods

public void SetupChart(string Title, string XaxisTitle, string Xaxis2Title, string YaxisTitle, double
XaxisMinimum, double Xaxis2Minimum, double XaxisMaximum, double Xaxis2Maximum)
{
Chart chartObj = this.FindName("XvsYChart") as Chart;

if (chartObj != null)
{
chartObj.Name = "XvsYChart";
this.Title = Title;
this.TitleXAxis = XaxisTitle;
this.TitleXAxis2 = Xaxis2Title;
```

```
this.TitleYAxis = YaxisTitle;

foreach (Series series in chartObj.Series)
{
series.ClearPoints();
}

var defaultArea = chartObj.ChartAreas[DEFAULT_CHART_AREA];
{
// Reset the zoom.
defaultArea.AxisX2.ScaleView.ZoomReset();
defaultArea.AxisY.ScaleView.ZoomReset();

defaultArea.AxisX2.Minimum = Xaxis2Minimum;
defaultArea.AxisX2.Maximum = Xaxis2Maximum;
defaultArea.AxisX2.Interval = (defaultArea.AxisX2.Maximum - defaultArea.AxisX2.Minimum) / 10;

defaultArea.AxisY.Minimum = double.NaN;
defaultArea.AxisY.Maximum = double.NaN;
defaultArea.AxisY.Interval = (defaultArea.AxisY.Maximum - defaultArea.AxisY.Minimum) / 10;

if (!defaultArea.Visible)
defaultArea.Visible = true;
}

var markerArea = chartObj.ChartAreas[Marker_CHART_AREA];
{
// Reset the zoom.
markerArea.AxisX.ScaleView.ZoomReset();
markerArea.AxisY.ScaleView.ZoomReset();

markerArea.AxisX.Minimum = XaxisMinimum;
markerArea.AxisX.Maximum = XaxisMaximum;
markerArea.AxisX.Interval = (markerArea.AxisX.Maximum - markerArea.AxisX.Minimum) / 10;

markerArea.AxisY.Minimum = 0;
markerArea.AxisY.Maximum = 1;
markerArea.AxisY.Interval = 1;

if (!markerArea.Visible)
markerArea.Visible = true;
}
```

```
}
}

public void DrawSeries(double[] xValues, double[] results, int seriesIndex, AxisType axisType)
{
var chart = this.FindName("XvsYChart") as Chart;
var series = chart.Series[seriesIndex];

series.ClearPoints();
series.Points.SuspendUpdates();
series.XAxisType = axisType;
for (int i = 0; i < xValues.Length; i++)
{
series.Points.AddXY(xValues[i], results[i]);
}
series.Points.ResumeUpdates();

enableSeries(series, true);
}

#endregion Public Methods

#region Private Methods

/// <summary>
/// Method called when the mouse is clicked. Determines if the click was on the MS Chart Legend and then checks / unchecks the Site selection.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void XvsYChart_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
{
try
{
Chart myPowerVsPoint = this.FindName("XvsYChart") as Chart;

System.Windows.Forms.DataVisualization.Charting.HitTestResult result =
myPowerVsPoint.HitTest(e.X, e.Y);

if (result != null && result.Object != null)
{
```

```
// When user hits the LegendItem
if (result.Object is LegendItem)
{
// Legend item result
LegendItem legendItem = result.Object as LegendItem;

// series item selected
Series selectedSeries = legendItem.Tag as Series;

// toggle enabled
enableSeries(selectedSeries, !selectedSeries.Enabled);

}
}
}
catch (Exception ex)
{
MessageBox.Show("An error occured : " + ex.Message, "Merlin Test Studio");
}

}

private void enableSeries(Series series, bool enabled)
{
if (series.Enabled != enabled)
{
series.Enabled = enabled;
(series.Tag as LegendItem).Cells[0].Image = enabled ? IMAGE_NAME_CHECKED :
IMAGE_NAME_UNCHECKED;
}
}

#endregion Private Methods

#region Dependency Properties

public static readonly DependencyProperty ChartTypeProperty =
DependencyProperty.Register("ChartType", typeof(SeriesChartType), typeof(XvsYGenericChart),
new FrameworkPropertyMetadata
{
BindsTwoWayByDefault = true,
});
```

```
/// <summary>
/// Gets / Sets the chart type to use.
/// </summary>
public SeriesChartType ChartType
{
get
{
return (SeriesChartType)GetValue(ChartTypeProperty);
}
set
{
Chart myPowerVsPoint = this.FindName("XvsYChart") as Chart;

foreach (var series in myPowerVsPoint.Series)
{
series.ChartType = value;
//if (value == SeriesChartType.FastPoint)
//{
//    series.MarkerSize = 3;
//    series.MarkerStyle = MarkerStyle.Cross;
//}
}

SetValue(ChartTypeProperty, value);
}
}

#endregion Dependency Properties
}
}
```