Workspace Desktop Edition Developer's Guide

Configure Your Custom Theme

4/30/2025

Configure Your Custom Theme

As detailed in Branding section of the Developer's Guide, you can easily add your own branding to Workspace Desktop Edition.

In 8.5, you can also create your own custom theme, with new colors and icons, as detailed in the following sections.

Add a New Theme

To add a new theme to the Workspace Desktop Edition, add your theme to the <themes> section of a configuration file.

- Best practice: Define a theme in a plugin configuration file; for instance, in the Custom Theme of the Workspace Desktop Edition sample, you can define your theme in the `Genesyslab.Desktop.Modules.CustomThemeSample.module-config` file.
  or
- Add your theme to the <themes> section in the `InteractionWorkspace.exe.config` file .

For each new theme, the tag <theme> includes the list of dictionaries to load for the theme:

- The <theme> tag include three properties:
  - `name`: The name of theme, which must be unique.
  - `displayNameKey`: The entry key defined in the language dictionaries and used to display menu items.
  - `mainResourceDictionary`: Path of the XAML resources used in the dictionaries (styles and colors).

> **Important**
> The attribute values for `displayNameKey` and `mainResourceDictionary` can be overriden.

- The <xmlDictionaries></xmlDictionaries> tags include the list of XML dictionaries which contain all the paths for icons and images. Its <xmlDictionary></xmlDictionary> subtags each define an XML dictionary with the two following attributes:
  - `name`: The name of the dictionary.
  - `path`: The path to the dictionary file.

The sample themes are available in the Workspace Desktop Edition's installation Package. The following XML code shows the list of default themes

delivered with the Workspace Desktop Edition:

```
<themes>
     <!-- Default theme -->
    <theme name="Default" displayNameKey="Theme.Default.DisplayName" xamlDictionary="/Genesyslab.Desktop.WPFCommon;component/themes/
generic.xaml">
      <xmlDictionaries>
        <xmlDictionary name="iw"  path=".\Resources\ResourcesDefinition85.xml"></xmlDictionary>
      </xmlDictionaries>
    </theme>
    <!-- Additional themes -->
    <theme name="Royale" displayNameKey="Theme.Royale.DisplayName" mainResourceDictionary="/Genesyslab.Desktop.WPFCommon;component/themes/
royal.xaml">
       <xmlDictionaries>
        <xmlDictionary name="iw"  path=".\Resources\ResourcesDefinition.xml"></xmlDictionary>
      </xmlDictionaries>
    </theme>
    <theme name="Fancy" displayNameKey="Theme.Fancy.DisplayName" mainResourceDictionary="/Genesyslab.Desktop.WPFCommon;component/themes/
fancy.xaml">
       <xmlDictionaries>
        <xmlDictionary name="iw"  path=".\Resources\ResourcesDefinition.xml"></xmlDictionary>
      </xmlDictionaries>
    </theme>
  </themes>
```

You can extend and override the values or XML dictionaries, as shown below in the Custom Theme Sample.

```
<themes>
    <!-- Extend and Override the Default theme -->
   <theme name="Default" displayNameKey="Theme.MyDefault.DisplayName"><!-- override displayNameKey -->
      <xmlDictionaries>
        <xmlDictionary name="rebranding"   path=".\Resources\RebrandingTheme.xml"></xmlDictionary> <!-- Extend the XML Dictionaries-->
      </xmlDictionaries>
    </theme>

    <!-- Add a new Custom Theme -->
   <theme name="CustomTheme" displayNameKey="Theme.Custom.DisplayName"
mainResourceDictionary="/Genesyslab.Desktop.Modules.CustomThemeSample;component/Resources/themes/CustomTheme.xaml">
      <xmlDictionaries>
        <xmlDictionary name="iw"  path=".\Resources\ResourcesDefinitionCustom.xml"></xmlDictionary>
      </xmlDictionaries>
    </theme>
```

```
  </themes>
```

The `CustomTheme.Xaml` file must declare the main resource dictionary of the new Workspace Edition style in addition to the Custom Color dictionary.

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <ResourceDictionary.MergedDictionaries>
    <!-- New IW Style -->
    <ResourceDictionary Source="/Genesyslab.Desktop.WPFCommon;component/Resources/NewStyles/NewStylesResourceLibrary.xaml"/>
    <ResourceDictionary Source="/Genesyslab.Desktop.Modules.CustomThemeSample;component/Resources/ColorBrushes/
CustomDefaultColorTheme.xaml"/>
  </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

You can also set up the theme at the Workspace Desktop Edition startup (SplashScreen, First Login Panel, and so on) by adding the `gui.theme` key to the `appSettings` section of the `InteractionWorkspace.exe.config` configuration file.

```
<appSettings>
       <add key="gui.theme" value="Default"/><!-- Name of theme-->
</appSettings>
```

Customize the Interaction Bar Framework

The Workspace Desktop Edition implements the same Interaction Toolbar for the Interaction Bar and Bundle Bar areas with different colors; the Interaction Bar's components are dark and the Bundle Bar's components are light.

You can customize the Interaction Toolbar without creating a new one. You can set up several buttons associated with the following components by setting the the `ButtonStyle` dependency property with the following values:

- `InteractionToolBarButton`
- `InteractionToolBarSplitButton`
- `InteractionToolBarToggleButton`
- `InteractionToolBarDropDownButton`

The following XAML sample shows how to customize the style of the `ButtonSingleStepTransfer` component.

```
<commonControls:InteractionToolBarButton localization:Translate.Uid="Windows.VoiceView.ButtonSingleStepTransfer" ButtonStyle="{Binding
ButtonStyle}"
            Name="buttonSingleStepTransfer"
            Click="SingleStepTransferCommand_Click"
            ToolTip="{Binding Path=ToolTipButtonSingleStepTransfer, ElementName=callView}"
            AutomationProperties.Name="{Binding Path=PropertiesNameButtonSingleStepTransfer, ElementName=callView}">
      <commonControls:InteractionToolBarButton.Visibility>
            <MultiBinding Converter="{StaticResource visibilityBooleanORConverter}" Mode="OneWay" FallbackValue="Collapsed">
                <Binding Path="Interaction.IsItPossibleToOneStepTransfer"  FallbackValue="Visible" Mode="OneWay"
Converter="{StaticResource visibilityConverter}"/>
                <Binding Path="Interaction.IsItPossibleToCompleteTransferActiveConsultation"  FallbackValue="Visible" Mode="OneWay"
                         Converter="{StaticResource visibilityConverter}" />
            </MultiBinding>
        </commonControls:InteractionToolBarButton.Visibility>
        <StackPanel Orientation="Horizontal">
                <commonControls:MagicImage localization:Translate.Uid="Common.Images.Interaction.Voice.Transfer"
Source="{localization:Translate}"
                          ResourceKey="{localization:Translate}" Width="24" Height="24" RenderOptions.BitmapScalingMode="NearestNeighbor"
/>
                <Path VerticalAlignment="Center" Margin="3" Data="M0,0L3,3 6,0z"  Width="{Binding Source=8, Converter={StaticResource
relativeSizeConverter}}"
                           Height="{Binding Source=4, Converter={StaticResource relativeSizeConverter}}"
                           Fill="{Binding Foreground, ElementName=buttonSingleStepTransfer}">
                </Path>
        </StackPanel>
</commonControls:InteractionToolBarButton>
```

The view model must also implement the `IButtonStyle` interface.

```
ButtonStyle buttonStyle;
public ButtonStyle ButtonStyle
{
       get { return buttonStyle; }
       set
       {
            if (buttonStyle != value)
            {
                  buttonStyle = value;
```

```
                    OnPropertyChanged("ButtonStyle");
                }
            }
}
```

A helper provides the view context's information about the toolbar style to the `ModelView`, as shown below.

```
public void Create()
{
          IDictionary<string, object> contextDictionary = Context as IDictionary<string, object>;
          HelperToolbarFramework.SetButtonStyle(contextDictionary, Model);
}
```