Wisej Themes

Theme System

1 Overview

Wisej theme system is quite sophisticated and goes beyond simple CSS or SASS. This document is only a short overview to get you started. The full documentation will be ready at some point during the beta period and before the final release.

In a nutshell, themes in Wisej are single JSON files that define colors, images, fonts, and appearances. Appearances are group of styles and properties organized by a key name and by state (default, pushed, hovered, horizontal, …) that are applied to the widgets that decide to use a specific appearance.

For example, the Wisej.Web.Button control naturally uses the "button" appearance. In the JSON file "appearances\button" is a map of "states", each state holds "styles" and "properties".

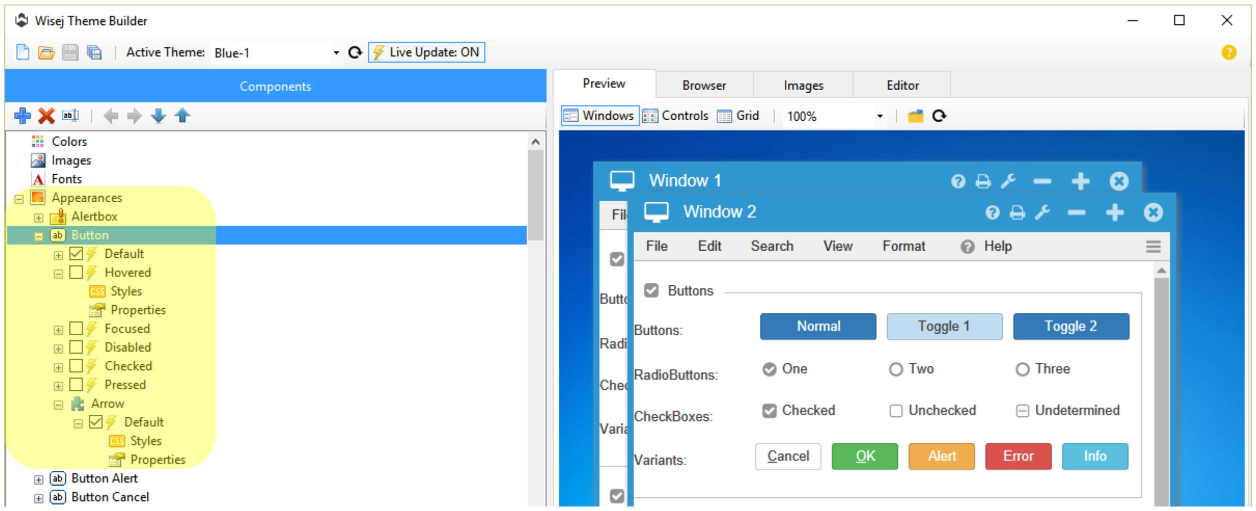
Styles correspond to CSS rules (classes) while Properties correspond to actual properties on the JavaScript widget. While styles are limited by CSS, properties are unlimited and depend on the widget itself.

1.1 Child Components

Wisej JavaScript widgets are usually complex widgets containing other widgets (e.g. the button widget contains an image widget and a label widget) and the theme system has to be able to style the main widget and all its children since they are an integral part of the widget.

Child components are themed as "components" in the parent appearance, each component contains "styles" and "properties" and more "components". This recursion can go very deep.

For example, to theme the image component in a button, the appearance path is "\appearances\button\components\icon".



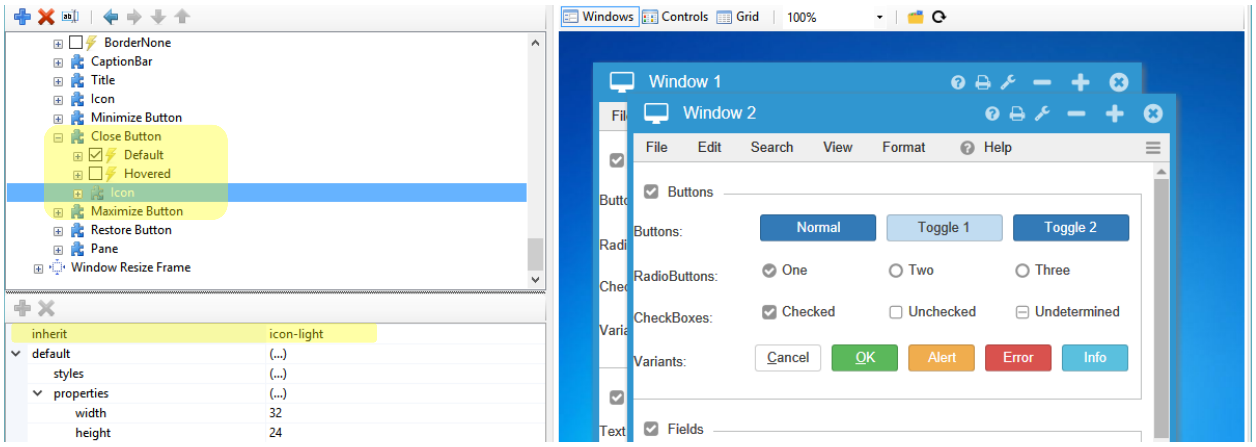
1.2 Inheritance

Many styles and properties set on a specific appearance would do well on others, especially when a widget (like the label for example, or the image) is used many times as a child widget to compose more complex controls. In fact, it is not necessary to theme a child widget if its default appearance is good enough.

To simplify the definition and manageability of the themes, Wisej support inheritance between appearances. A child component can specify that it wants to inherit all the styles and properties from another appearance and simply override or add some.

The inheritance system can also be used to indirect certain theme features. The same way that images, colors and fonts are all indirected.

In the themes that we provide, for example, we have defined two appearances named “icon-dark” and “icon-light” with certain styles and properties, then whenever a widget has child component that is an icon, it can simply set the “inherit” value to one of those two. When creating a new theme, we can simply change the theme of those two to change the entire theme.



inherit	icon-light
default	(...)
styles	(...)
properties	(...)
width	32
height	24

1.3 Custom Theming

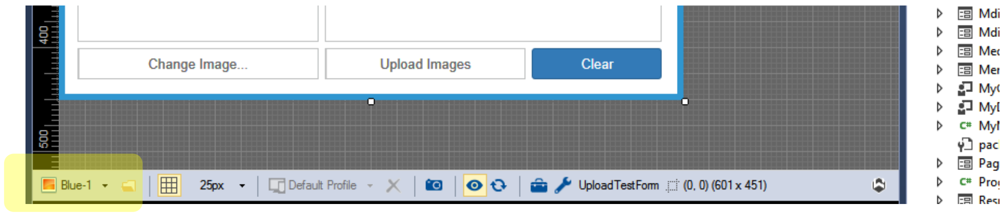
Because of the extreme flexibility of the theme system, Wisej is capable of theming specific element in an application. All Wisej controls expose a property “AppearanceKey”, you can change that property to change the theme for a specific component. Then add a new appearance to the theme and you are now able to theme a specific part of your UI.

2 How to Use the Additional Themes

The additional themes that we provide, or any custom theme that you build, can be used quite easily simply by loading the theme file in the designer, to see it at design time, and in Web.config to select the theme at runtime.

2.1 Use a Custom Theme at Design Time

With the designer open, you can select the theme using the list control in our designer strip.

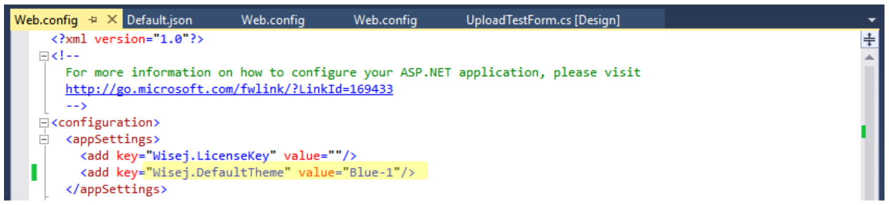


The list automatically includes: all the build-in themes + all the *.theme files found in the \Themes directory in the current project. To load a theme not in the list, click the folder button to select any theme file from any location.

Note that the theme that you select in the designer is only used by the designer. It will not change the theme used at runtime.

2.2 Set the Default Runtime Theme

You can change the default theme used at runtime in Web.config using the “Wisej.DefaultTheme” application setting



```xml
<?xml version="1.0"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->
<configuration>
  <appSettings>
    <add key="Wisej.LicenseKey" value=""/>
    <add key="Wisej.DefaultTheme" value="Blue-1"/>
  </appSettings>
```

Change the value using only the theme name, without the extension or path. Wisej will load the theme from the embedded resources or the \Themes path in your application.

Once the theme is changed in Web.config it will also be the default theme when you open a new designer.

Default Theme Override

Wisej uses the default theme – the one specified in Web.config – when the property “theme” in the application configuration JSON file is not set.

If you add the “theme” property to Default.json or {ApplicationName}.json in case you have multiple sub-applications in the same project, that is the theme that Wisej will use, overriding the default setting from Web.config.

2.3 Change the Theme at Runtime

To change the theme at runtime, use the Application.LoadTheme() method. Unlike most other web theme systems, Wisej can change the theme dynamically without reloading the page. It will seamlessly recalculate all the borders, non-client areas, update the images, etc.

In your code, when you want to change the theme, simply call:

```
// Clear-2 can be changed to the name of your theme.
Application.LoadTheme(“Clear-2”);
```

3 Theme Mixins

Mixins are a common concept used in JavaScript to merge a class “on top” of another class to add shared properties and methods. A class mixin is like an “overlay” on top of a class.

We have applied the same concept to themes. Why?

Usually any theme system (ASP.NET, jQuery, or any CSS or SASS based system) defines a core set of styles for a defined set of components in a core list of themes. In our case we provide 4 built-in themes and several additional downloadable themes.

However, if you want to add or change something to the theme - add a new component, or change a style or property – you have to duplicate a base theme of your choice and change or add what you need. With themes limited to CSS you may be able to override certain rules but it quickly becomes very complicated to pin point all the things to override.

In any case, once the root theme is duplicated and extended or modified you are stuck with the root theme.

Another scenario, very important for Wisej because of its open nature to integrations, is the development of extensions. Some new components may also need to be themed (see the Wisej.Web.Ext.Bubble example) but the extension developer cannot go back and modify all the existing themes and the future themes to add its own styles and properties.

Here is where theme mixins are a great solution. The extension developer has to create only one partial theme specific to the new component and name it “{theme name}.mixin.theme” and place it as an embedded resource in the project.

Wisej will automatically load all the mixin theme files and merge them with the selected base theme every time the application loads a theme, at design time too!

A theme mixin can extend a theme – add a new appearance, or color, or image – and it can also override anything in the base theme, **providing unmatched flexibility in Wisej Theme System.**