EPiSERVER WORLD

Register New password Login

Blogs / 2014 / 4 /

Strongly typed TinyMCE settings

Tuesday, 08 April 2014

By: Torjus Eidet

Number of votes: 4
Views: 1294
Average rating: ■■■■■

Syndication and Sharing

One of the things that bugs me when migrating between environments is to set up the TinyMCE settings. Wouldn't it be great if we could just define it in code, like we do with our content types? It's actually not that hard to accomplish.

With this solution, all you have to do is to decorate you XHtmlString properties with an attribute, TinyMceSettings.

```
    public class ExamplePage : PageData
    {
        [Display(Name = "Main body")]
        [TinyMceSettings(typeof(StandardTinyMceSettings))]
        public virtual XhtmlString MainBody { get; set; }
    }
```

This attribute takes the type of a settings class as it parameter.

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
    public class TinyMceSettingsAttribute : Attribute
    {
        public TinyMceSettingsAttribute(Type settingsType)
        {
            SettingsType = settingsType;
        }
        public Type SettingsType { get; set; }
    }
```

A settings class looks like this:

```
    public class StandardTinyMceSettings : ITinyMceSettings
    {
        public StandardTinyMceSettings()
        {
            DisplayName = "Some TinyMce Settings";
            Id = new Guid("2235405F-A998-4371-81AE-F45F8899A03A");
            ContentCss = null;
            Toolbars = new []
            {
                new []
                {
                    "bold", "italic", "underline", "strikethrough", "separator",
                    "justifyleft", "justifycenter", "justifyright", "separator",
                    "epilink", "unlink"
                }
            };
            NonVisualPlugins = null;
        }

        public string DisplayName { get; set; }
        public Guid Id { get; set; }
        public string ContentCss { get; set; }
        public string[][] Toolbars { get; set; }
        public string[] NonVisualPlugins { get; set; }
    }
```

As you can see, you can define toolbars, display name, non visual plugins, editor css file path and so on. In order for this solution to work, you will need to implement the interface ITinyMceSettings:

```
    public interface ITinyMceSettings
    {
        string DisplayName { get; set; }
        Guid Id { get; set; }
        string ContentCss { get; set; }
        string[][] Toolbars { get; set; }
        string[] NonVisualPlugins { get; set; }
    }
```

In order to hook things up you will need to include the following initializable module:

```
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    [InitializableModule]
    public class TinyMceSettingsInitialization : IInitializableModule
    {
        private bool _initialized;
        IPropertySettingsRepository _propertySettingsRepository;
        IContentTypeRepository _contentTypeRepository;
        IPropertyDefinitionRepository _propertyDefinitionRepository;

        public void Initialize(InitializationEngine context)
        {
            if (_initialized) return;
            _propertySettingsRepository = ServiceLocator.Current.GetInstance<IPropertySettingsRepository>();
            _contentTypeRepository = ServiceLocator.Current.GetInstance<IContentTypeRepository>();
            _propertyDefinitionRepository = ServiceLocator.Current.GetInstance<IPropertyDefinitionRepository>();


            var allContentTypes = _contentTypeRepository.List();
            foreach (var contentType in allContentTypes)
            {
                if (contentType == null || contentType.ModelType == null)
                {
                    continue;
                }

                var properties = contentType.ModelType.GetProperties();
                foreach (var propertyInfo in properties)
                {
                    if (propertyInfo.PropertyType != typeof (XhtmlString))
                    {
                        continue;
                    }

                    var settings = GetSettingsFromAttrubte(propertyInfo);
                    if (settings == null)
                    {
                        continue;
                    }

                    CreateOrUpdateSettingsContainer(settings);

                    var property = contentType.PropertyDefinitions.First(x => x.Name == propertyInfo.Name);

                    SaveSettingsToProperty(property, settings);
                }
            }

            _initialized = true;
        }

        public ITinyMceSettings GetSettingsFromAttrubte(PropertyInfo propertyInfo)
        {
            var settingsAttribute = (TinyMceSettingsAttribute)Attribute.GetCustomAttribute(propertyInfo, typeof(TinyMceSettingsAttribute));
            if (settingsAttribute == null) return null;

            var settingsType = settingsAttribute.SettingsType;
            var settings = Activator.CreateInstance(settingsType) as ITinyMceSettings;
            if (settings == null) throw new Exception("Defined TinyMceSettings type is not implementing ITinyMceSettings");

            return settings;
        }

        public void CreateOrUpdateSettingsContainer(ITinyMceSettings settings)
        {
            PropertySettingsContainer container;
            _propertySettingsRepository.TryGetContainer(settings.Id, out container);
            container = container ?? new PropertySettingsContainer(settings.Id);

            var wrapper = container.GetSetting(typeof(TinyMCESettings));
            wrapper = wrapper ?? new PropertySettingsWrapper();

            var propertySettings = new TinyMCESettings();
            foreach (var toolbarRow in settings.Toolbars)
            {
                propertySettings.ToolbarRows.Add(new ToolbarRow(toolbarRow));
            }
            propertySettings.NonVisualPlugins = settings.NonVisualPlugins ?? new string[] { };
            propertySettings.ContentCss = settings.ContentCss ?? string.Empty;

            wrapper.PropertySettings = propertySettings;
            wrapper.IsGlobal = true;
            wrapper.IsDefault = false;
            wrapper.DisplayName = settings.DisplayName;

            container.AddSettings(wrapper);

            _propertySettingsRepository.Save(container);
        }

        private void SaveSettingsToProperty(PropertyDefinition property, ITinyMceSettings settings)
        {
            var writableProperty = property.CreateWritableClone();
            writableProperty.SettingsID = settings.Id;
            _propertyDefinitionRepository.Save(writableProperty);
        }

        public void Uninitialize(InitializationEngine context) { }
        public void Preload(string[] parameters)  { }
    }
```

This is just my first thoughts around such a system. This might not be the best way to solve this, so if you have any suggestions please feel free to share them with me. My plans are to turn this into a nuget package, but I'm hoping for some feedback first.

Code: https://gist.github.com/torjue/1ad223a637a024859428

tinyMCE EPiServer 7.5 Extending EPiServer

Comments

(By Martin Pickering , 08 April 2014 23:46, Permanent link)

what a stupendous idea.
if you are truly frustrated by migrating between environments why would you not have a standard set of settings that apply by default to all xhtmlstrings
instead of forcing you to apply the "default" via an attribute class. you could still retain the attribute class to be able to request specific settings for a particular property. perhaps also another attribute class to request standard episerver behaviour (to reverse the new default behaviour created by the custom code).

(By Linus Ekström , 09 April 2014 08:13, Permanent link)

Hi Torjus!

Nice to see someone try doing a generic solution to this finally. As a conscience, I did a prototype for another solution for this last week that I'll show for
the EPiServer development team today. So there is a chance that there might be a built in solution for this in the core in the near future.

(By Erik Lidälv , 15 May 2014 14:14, Permanent link)

Linus, do you have any news if/when this will be included in the core product?

Best regards
Erik