Transition (Managed)

# Overview:

The user should be able to add transitions between two photos. Currently we only want to provide two simple transitions: Fade transition and Fly-in transition. We may expand the transition library in the future. So we need to design an interface for the transition feature.

This document only covers the managed part. See the corresponding spec for the native interface design.

# Goals:

* Create an interface for the application to interact with the transition library.

# None Goals:

* Create a plug-in model.
* Implement the actual transition library.

# Design:

We provide both an interface and an abstract base class that contains some default implementation of the interface. Whether to remove the interface and use the abstract base class only is to be debated. Anyway, the interface is named ITransition, and is designed as below.

    public interface ITransition

    {

        string Name { get; }

        TimeSpan TransitionDuration { get; set; }

        bool ImageZIndexModified { get; }

        // Foreground/BackgroundElement can be either Image or MediaElement.

        // Set these properties in the PreviewPage.

        FrameworkElement ForegroundElement { get; set; }

        FrameworkElement BackgroundElement { get; set; }

        event EventHandler TransitionCompleted;

        ITransition Clone();

        // Start/stop the transition.

        void Begin();

        void Stop();

        // Serilaize/deserialize the transition.

        void Save(XElement transitionElement);

    }

## Common properties/events/methods

A transition should have a unique name. The user can specify how much time to spend on transition for each photo. Currently we require each transition to perform on two photos: A background photo (the photo to be transited) and a foreground photo (the photo currently being displayed). The preview page is responsible for passing the correct background/foreground image to the transition. After a transition is completed, the TransitionCompleted event must fire to notify the preview page. The TransitionBase class has a default implementation that fires this event. Finally, the compose page requires each transition to support deep clone. Each transition can implement the Clone method, and choose which property (or properties) need(s) to be cloned.

## Z-Index

The preview page requires information about the background/foreground images’ z-index. Some transitions, such as the fade transition, do not change the z-index. Other transitions, such as the fly-in transition, change the z-index. For example, the fly-in transition requires the background image to cover the foreground image. In those cases, the transition must set ImageZIndexModified to true, so the preview page knows how to handle them.

## Start/stop the transition

The preview page requires to start/stop the transition. It delegates this task to the transition. Each transition must implement the Start/Stop methods. We will not provide default implementations in TransitionBase. If a transition uses a storyboard, it can simply begin/stop the storyboard.

## Serialization/deserialization

All transitions must support serialization/deserialization. We provide some common persistence logic in the TransitionBase class, while allow child classes to override it. TransitionBase implements Save method. It also contains a static Load method.

public static ITransition Load(XElement transitionElement);

The Load method should be static, as it creates an ITransition from an XElement. The design is still questionable as a seemingly better design is to create an empty ITransition and then invokes the instance Load method. The instance Load method should be exposed via the ITransition interface. But due to time constraint, we’ll leave it for now. Let’s review the design in next version.

TransitionBase also contains a LoadChild method for derived classes to perform further deserialization.