Changing a Layer's Default Behavior

Core Animation implements its implicit animation behaviors for layers using action objects. An action object is an object that conforms to the `CAAction` protocol and defines some relevant behavior to perform on a layer. All `CAAnimation` objects implement the protocol, and it is these objects that are usually assigned to be executed whenever a layer property changes.

Animating properties is one type of action but you can define actions with almost any behavior you want. To do that, though, you have to define your action objects and associate them with your app's layer objects.

Custom Action Objects Adopt the CAAction Protocol

To create your own action object, adopt the `CAAction` protocol from one of your classes and implement the `runActionForKey:object:arguments:` method. In that method, use the available information to perform whatever actions you want to take on the layer. You might use the method to add an animation object to the layer or you might use it to perform other tasks.

When you define an action object, you must decide how you want that action to be triggered. The trigger for an action defines the key you use to register that action later. Action objects can be triggered by any of the following situations:

- The value of one of the layer's properties changed. This can be any of the layer's properties and not just the animatable ones. (You can also associate actions with custom properties you add to your layers.) The key that identifies this action is the name of the property.
- The layer became visible or was added to a layer hierarchy. The key that identifies this action is `kCAOnOrderIn`.
- The layer was removed from a layer hierarchy. The key that identifies this action is `kCAOnOrderOut`.
- The layer is about to be involved in a transition animation. The key that identifies this action is `kCATransition`.

Action Objects Must Be Installed On a Layer to Have an Effect

Before an action can be performed, the layer needs to find the corresponding action object to execute. The key for layer-related actions is either the name of the property being modified or a special string that identifies the action. When an appropriate event occurs on the layer, the layer calls its `actionForKey:` method to search for the action object associated with the key. Your app can interpose itself at several points during this search and provide a relevant action object for that key.

Core Animation looks for action objects in the following order:

1. If the layer has a delegate and that delegate implements the `actionForLayer:forKey:` method, the layer calls that method. The delegate must do one of the following:
   - Return the action object for the given key.
   - Return `nil` if it does not handle the action, in which case the search continues.
   - Return the `NSNull` object, in which case the search ends immediately.
2. The layer looks for the given key in the layer's `actions` dictionary.

3. The layer looks in the `style` dictionary for an actions dictionary that contains the key. (In other word, the `style` dictionary contains an `actions` key whose value is also a dictionary. The layer looks for the given key in this second dictionary.)
4. The layer calls its `defaultActionForKey:` class method.
5. The layer performs the implicit action (if any) defined by Core Animation.

If you provide an action object at any of the appropriate search points, the layer stops its search and executes the returned action object. When it finds an action object, the layer calls that object's `runActionForKey:object:arguments:` method to perform the action. If the action you define for a given key is already an instance of the `CAAnimation` class, you can use the default implementation of that method to perform the animation. If you are defining your own custom object that conforms to the `CAAction` protocol, you must use your object's implementation of that method to take whatever actions are appropriate.

Where you install your action objects depends on how you intend to modify the layer.

- For actions that you might apply only in specific circumstances, or for layers that already use a delegate object, provide a delegate and implement its `actionForLayer:forKey:` method.
- For layer objects that do not normally use a delegate, add the action to the layer's `actions` dictionary.
- For actions related to custom properties that you define on the layer object, include the action in the layer's `style` dictionary.
- For actions that are fundamental to the behavior of the layer, subclass the layer and override the `defaultActionForKey:` method.

Listing 6-1 shows an implementation of the delegate method used to provide action objects. In this case, the delegate looks for changes to the layer's `contents` property and swaps the new contents into place using a transition animation.

Listing 6-1 Providing an action using a layer delegate object

```
- (id<CAAction>)actionForLayer:(CALayer *)theLayer
                        forKey:(NSString *)theKey {
    CATransition *theAnimation=nil;

    if ([theKey isEqualToString:@"contents"]) {

        theAnimation = [[CATransition alloc] init];
        theAnimation.duration = 1.0;
        theAnimation.timingFunction = [CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseIn];
        theAnimation.type = kCATransitionPush;
        theAnimation.subtype = kCATransitionFromRight;
    }
    return theAnimation;
}
```

Disable Actions Temporarily Using the CATransaction Class

You can temporarily disable layer actions using the `CATransaction` class. When you change the property of a layer, Core Animation usually creates an implicit transaction object to animate the change. If you do not want to animate the change, you can disable implicit animations by creating an

explicit transaction and setting its `kCATransactionDisableActions` property to `true`. Listing 6-2 shows a snippet of code that disables animations when removing the specified layer from the layer tree.

Listing 6-2 Temporarily disabling a layer's actions

```
[CATransaction begin];
[CATransaction setValue:(id)kCFBooleanTrue
                 forKey:kCATransactionDisableActions];
[aLayer removeFromSuperlayer];
[CATransaction commit];
```

For more information about using transaction objects to manage animation behavior, see Explicit Transactions Let You Change Animation Parameters.