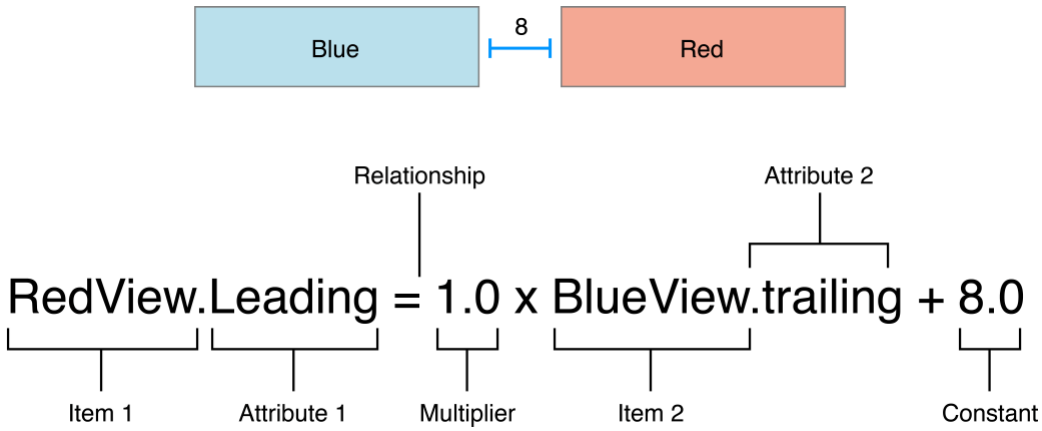
Changing Constraints

On This Page

A constraint change is anything that alters the underlying mathematical expression of a constraint (see Figure 17-1). You can learn more about constraint equations in Anatomy of a Constraint.

Figure 17-1 The constraint equation



All of the following actions change one or more constraints:

- Activating or deactivating a constraint
- Changing the constraint's constant value
- Changing the constraint's priority
- Removing a view from the view hierarchy

Other changes, like setting a control's property, or modifying the view hierarchy, can also change constraints. When a change occurs, the system schedules a deferred layout pass (see The Deferred Layout Pass).

In general, you can make these changes at any time. Ideally, most constraints should be set up in Interface Builder, or programatically created by the view controller during the controller's initial setup (for example, in `viewDidLoad`). If you need to dynamically change constraints at runtime, it's usually best to change them when the application's state changes. For example, if you want to change a constraint in response to a button tap, make that change directly in the button's action method.

Occasionally, you may need to batch a set of changes for performance reasons. For more information, see Batching Changes.

On This Page

The Deferred Layout Pass

Instead of immediately updating the affected views' frames, Auto Layout schedules a layout pass for the near future. This deferred pass updates the layout's constraints and then calculates the frames for all the views in the view hierarchy.

You can schedule your own deferred layout pass by calling the `setNeedsLayout` method or the `setNeedsUpdateConstraints` method.

The deferred layout pass actually involves two passes through the view hierarchy:

1. The update pass updates the constraints, as necessary
2. The layout pass repositions the view's frames, as necessary

Update Pass

The system traverses the view hierarchy and calls the `updateViewConstraints` method on all view controllers, and the `updateConstraints` method on all views. You can override these methods to optimize changes to your constraints (see Batching Changes).

Layout Pass

The system traverses the view hierarchy again and calls `viewWillLayoutSubviews` on all view controllers, and `layoutSubviews` (`layout` in OS X) on all views. By default, the `layoutSubviews` method updates the frame of each subview with the rectangle calculated by the Auto Layout engine. You can override these methods to modify the layout (see Custom Layouts).

Batching Changes

It is almost always cleaner and easier to update a constraint immediately after the affecting change has occurred. Deferring these changes to a later method makes the code more complex and harder to understand.

However, there are times when you may want to batch changes for performance reasons. This should only be done when char
changes.

On This Page

To batch a change, instead of making the change directly, call the `setNeedsUpdateConstraints` method on the view holding the constraint. Then, override the view's `updateConstraints` method to modify the affected constraints.

> NOTE
>
> Your `updateConstraints` implementation must be as efficient as possible. Do not deactivate all your constraints, then reactivate the ones you need. Instead, your app must have some way of tracking your constraints, and validating them during each update pass. Only change items that need to be changed. During each update pass, you must ensure that you have the appropriate constraints for the app's current state.

Always call the superclasses implementation as the last step of your `updateConstraints` method's implementation.

Do not call `setNeedsUpdateConstraints` inside your `updateConstraints` method. Calling `setNeedsUpdateConstraints` schedules another update pass, creating a feedback loop.

Custom Layouts

Override the `viewWillLayoutSubviews` or `layoutSubviews` methods to modify the results returned by the layout engine.

> IMPORTANT
>
> If possible, use constraints to define all of your layouts. The resulting layouts are more robust and easier to debug. You should only override the the `viewWillLayoutSubviews` or `layoutSubviews` methods when you need to create a layout that cannot be expressed with constraints alone.

When overriding these methods, the layout is in an inconsistent state. Some views have already been laid out. Others have not. You need to be very careful about how you modify the view hierarchy, or you can create feedback loops. The following rules should help you avoid feedback loops:

- You must
- You can safely invalidate the layout of views in your subtree; however, you must do this before you call the superclass's implementation.
- Don't invalidate the layout of any views outside your subtree. This could create a feedback loop.
- Don't call `setNeedsUpdateConstraints`. You have just completed an update pass. Calling this method creates a feedback loop.
- Don't call `setNeedsLayout`. Calling this method creates a feedback loop.
- Be careful about changing constraints. You don't want to accidentally invalidate the layout of any views outside your subtree.

On This Page