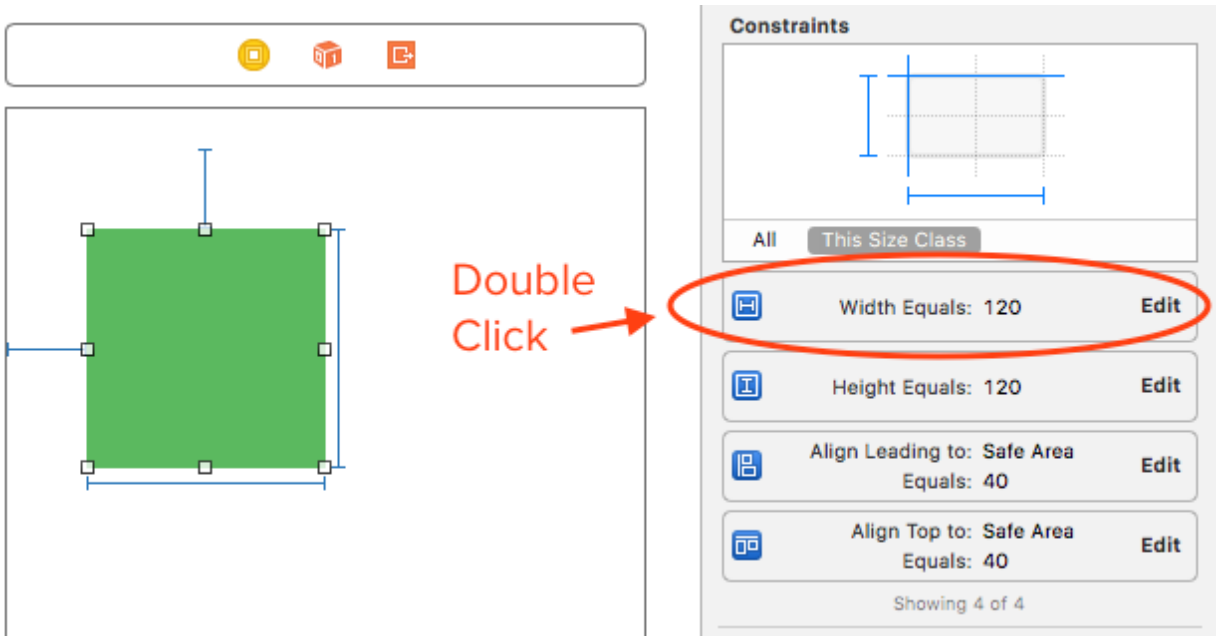
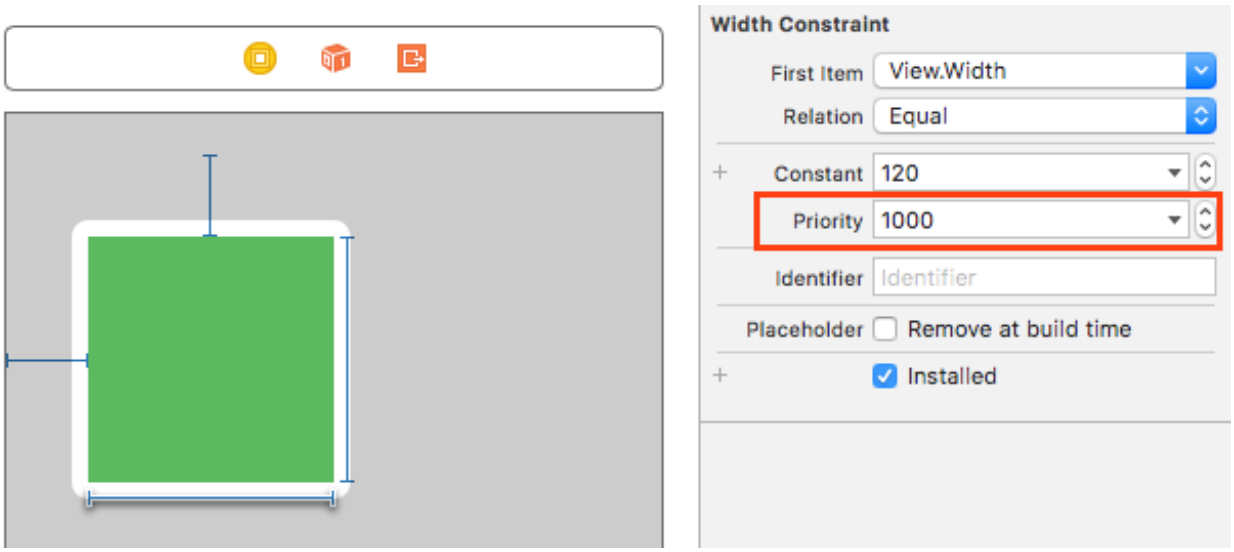
6 - What is constraint priority and how to use it

Previously, we have explained what is intrinsic content size and the usefulness of it. In this post we will look into what is constraint priority and how to use them.

To access constraint priority for an UI element, you can double click into the constraint.



You can see there's a field named "Priority", this is the priority of a constraint.



Constant priority is a number to determine how important is that constraint. The number can range from 1 to 1000, the higher the number goes, the more important a constraint is.

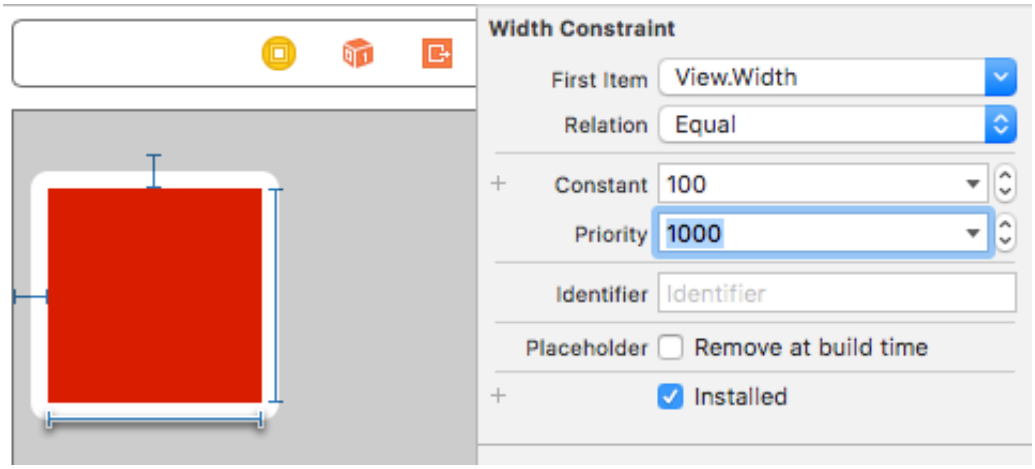
What does **important** mean? When there are conflicting constraints, Auto Layout Engine will attempt to break / ignore constraint with lower priority number first, hence making constraint with higher priority having higher importance.

If there are two conflicting constraints with the same priority number (making it a tie), Xcode will show you red lines with numbers.

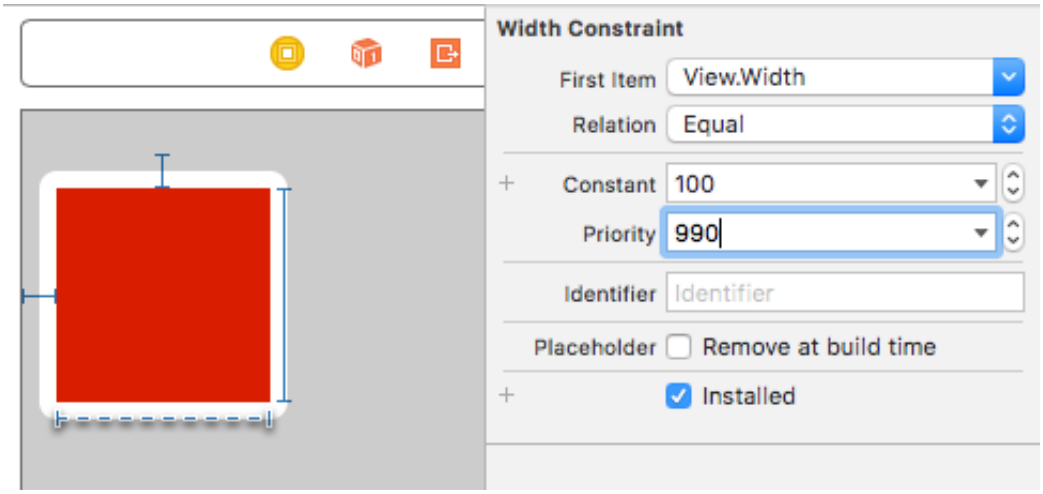
When you define a new constraint in Interface Builder / Storyboard, Xcode will auto assign the constraint priority as 1000. 1000 is the highest constraint priority, meaning it is **mandatory** as no other constraint can have higher priority than this and this constraint will never be ignored in favor of others.

On the other hand, constraints with priority less than 1000 (1-999) are called as **optional constraint** as it might be ignored in favor of other constraint which have higher constraint priority.

When you normally create a constraint (Xcode default set it to 1000 priority, mandatory constraint), it will show as a blue solid line like this :



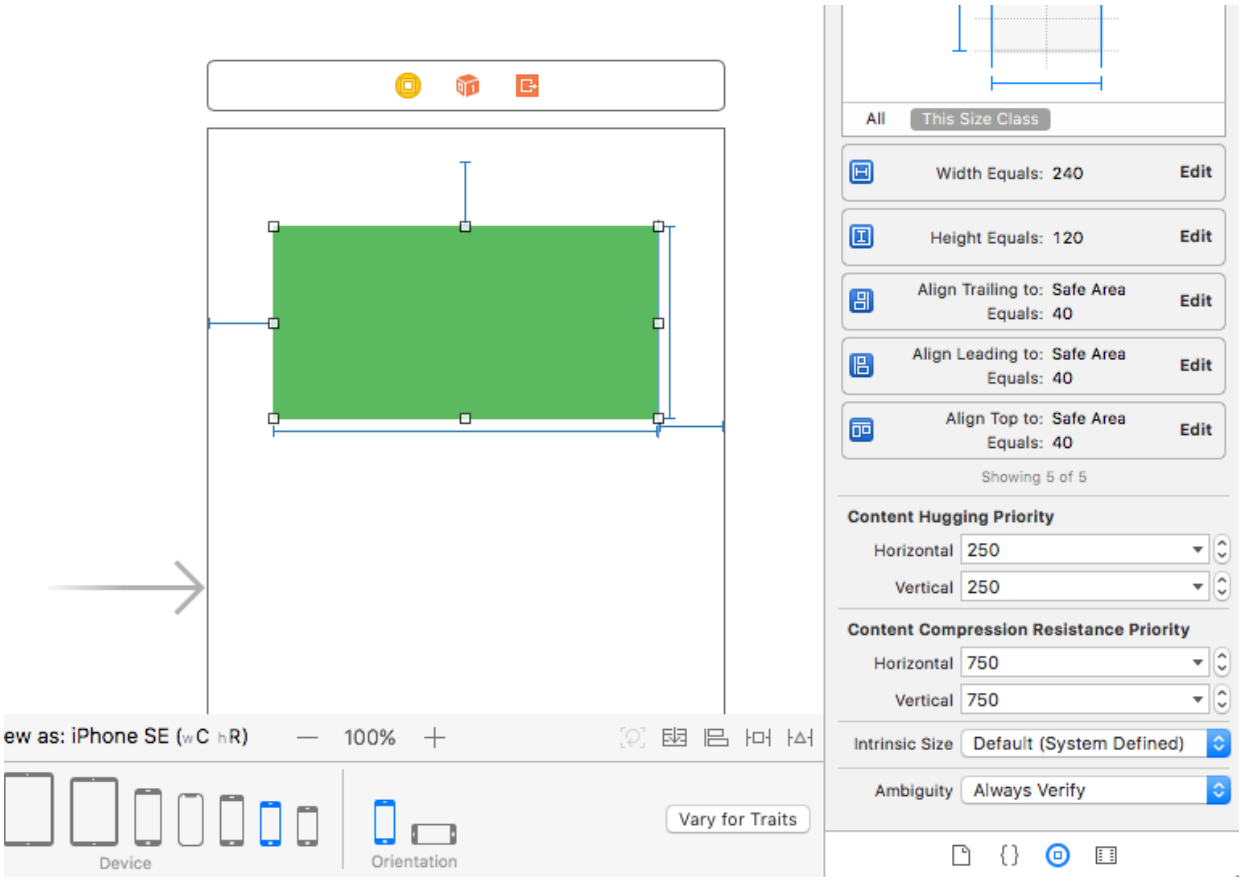
If you change the constraint priority to any number lower than 1000 (optional constraint), it will show dashed line like this :


Width Constraint
First Item
View.Width
Relation
Equal
Constant
100
Priority
990
Identifier
Identifier
Placeholder
Remove at build time
Installed

Example

Let's use an example similar to the one we used in the chapter "why conflicting constraint happen".

Here's a green view with some constraints defined, it works good on iPhone SE :



Its constraints are :

1. The width of green view is 240
2. The height of green view is 120
3. Leading space from screen left to left of the green view is 40
4. Trailing space from screen right to right of the green view is 40
5. Top space from screen top to top of the green view is 40

If you have read the chapter "Why conflicting constraint happen" previously, you know that constraint 1 and 3,4 will be conflicting on another screen size.

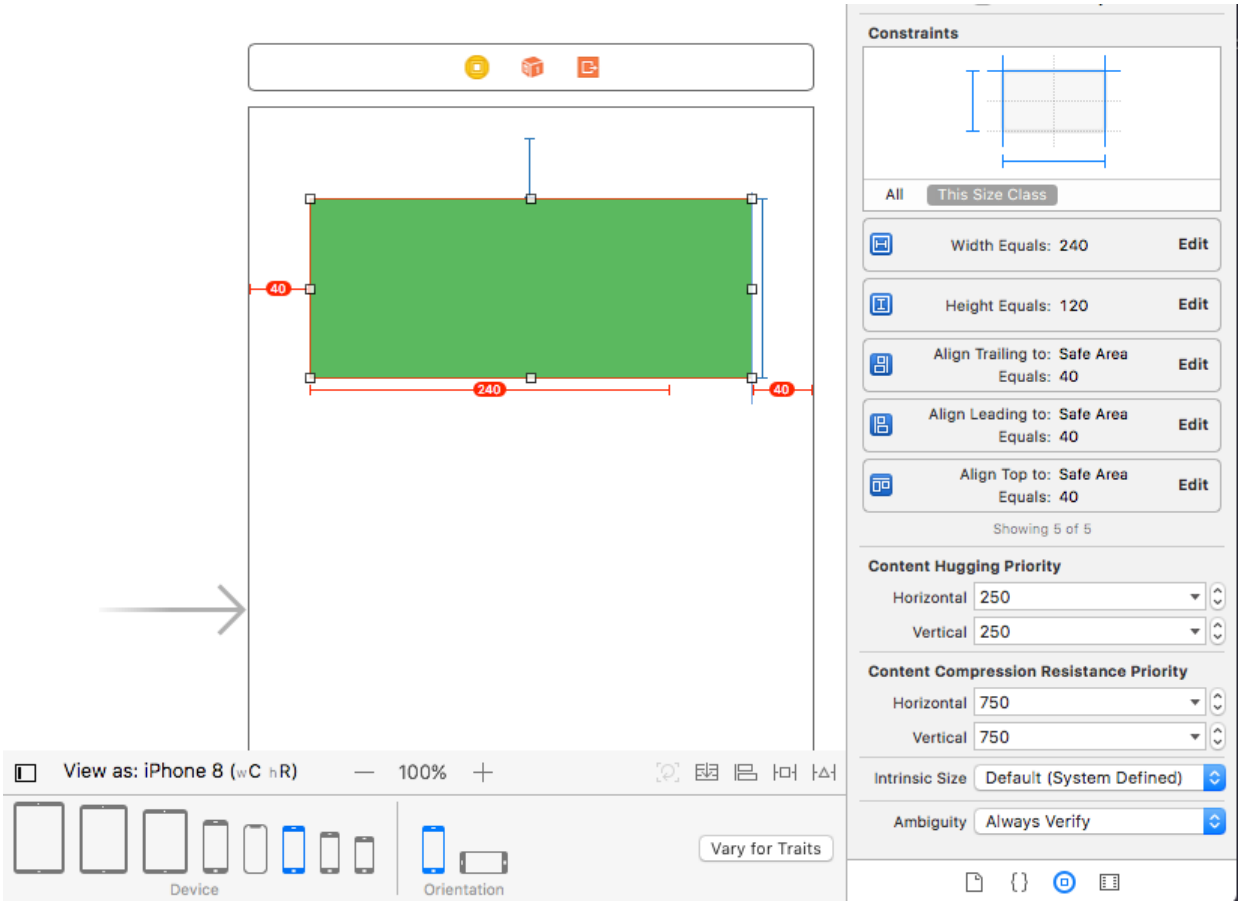
In iPhone SE, its screen width is 320 pt. Auto Layout Engine will use the screen width minus leading space and trailing space to calculate the width of the green view :

```
// iPhone SE
screenWidth = 320
leadingSpace = 40
trailingSpace = 40

greenViewWidth = screenWidth - leadingSpace - trailingSpace
greenViewWidth = 320 - 40 - 40
greenViewWidth = 240
```

Using the calculation above, Auto Layout Engine calculated that the width of green view should be 240. Then we also have another constraint explicitly specified the width of green view to 240, since these two values are equal, Xcode doesn't get confused and show blue lines.

But when we view this layout in iPhone 8 size, Xcode will complain that theres conflicting constraints :



It's conflicting because iPhone 8 screen width is 375. By using the leading and trailing constrains defined, Auto Layout Engine calculated that the width of green view :

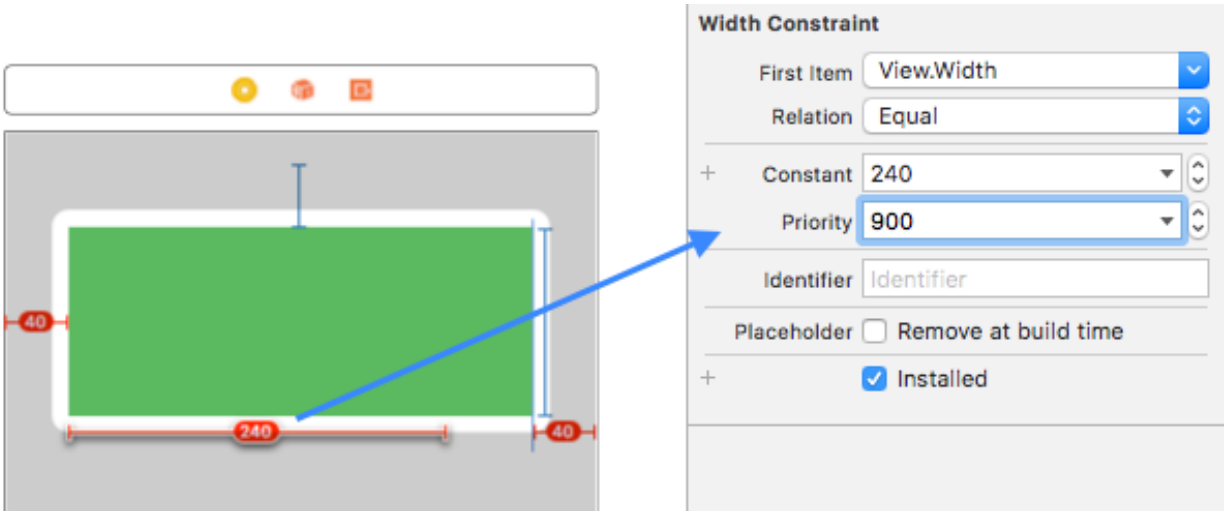
```
// iPhone 8
screenWidth = 375
leadingSpace = 40
trailingSpace = 40

greenViewWidth = screenWidth - leadingSpace - trailingSpace
greenViewWidth = 375 - 40 - 40
greenViewWidth = 295
```

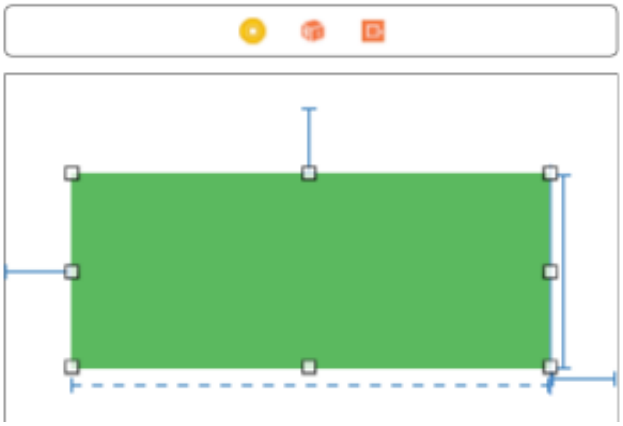
By calculation, width of green view will be 295, but at the same time there's also another constraint explicitly defined that the width of green view should equal to 240. Its impossible to make the green width to be both 295 and 240 at the same time, hence Xcode complains with red lines with numbers.

Notice that all the constraints are solid line, meaning they are mandatory and all have the same constraint priority (1000). Previously, we solved this by removing the explicit width constraint, now we can use constraint priority to solve this.

Lets set the priority of the explicit width constraint to a lower value, say 900.

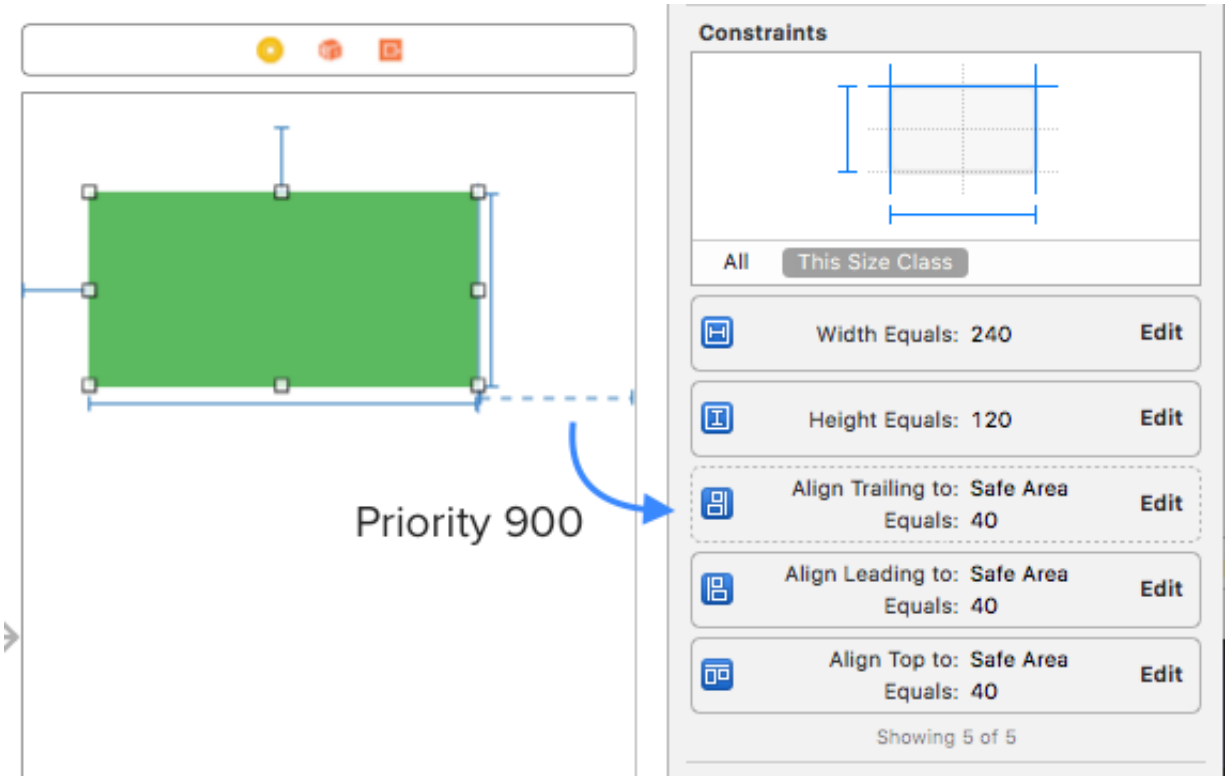


This will solve the conflicting constraint so Xcode will show blue lines :



Since the constraint priority of the explicit width constraint is 900 now, the leading constraint and trailing constraint has higher priority (1000). Thus when there's two possible width value, Auto Layout Engine will pick the one derived from constraints with higher priority, in this case, the leading and trailing constraint. The constraint with lower priority will be ignored by Auto Layout Engine if conflicting situation happens.

Similarly, we can keep the explicit width constraint priority to 1000 and instead lower the trailing constraint priority to 900.

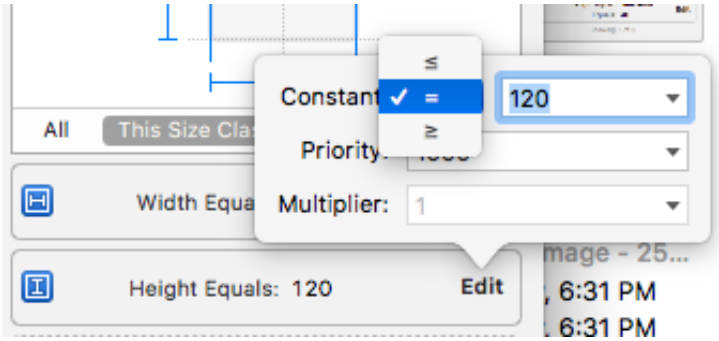


This will make the explicit width constraint priority higher than the trailing constraint, thus Auto Layout Engine will use the explicit width constraint as the width of the green view and ignore the trailing constraint (and also the calculation of width by using leading/trailing constraint). Doing this will make the green view width become 240 for all screen sizes.

Summary

Constraint priority is useful when you have possible conflicting constraint and you want to tell Xcode which of them can be safely ignored when conflict arise in order to solve conflict. Usually you will want to modify constraint priority of a constraint which uses **larger or equal to** (>=) / **smaller or equal to** (<=) instead of plain "equal" (==).

eg: You can set a constraint that specify the view height must be larger or equal to 120 pt.



Enjoyed this series?

Thanks for taking the time to learn about the fundamentals of Auto Layout! I hope this series has helped you on understanding why layout error happens and how to solve them.

Congratulations for making it this far! I wrote this series at around March 2018, I have since compiled, remastered this series into a book along with additional chapters on application of Auto Layout such as

1. Using ScrollView with dynamic content fetched from Web API
2. How to create a table view cell with dynamic height (content fetched from Web API, eg: Reddit app comment cell)
3. How to use Stack View to ease effort on creating constraint
4. Animating constraint etc.

The compiled book also include few Xcode projects (Xcode 9.2+) which you can follow along while going through the chapters so you can have some hands on experience too.

As a gratitude that you've spent effort and time to complete this series, you can use the promo code **series25** on the checkout page to get 25% off the compiled book!

Grab your copy of **Making Sense of Auto Layout** book