07. Intro to Autolayout

CSS DONE PROPERLY

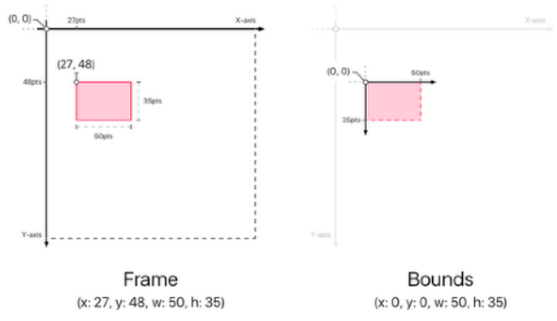
neverstop

INFINUM

01

Coordinate system

Frames



Frame
(x: 27, y: 48, w: 50, h: 35)

Bounds
(x: 0, y: 0, w: 50, h: 35)

- **frame** = a view's location and size using the **parent view's coordinate system** (important for placing the view in the parent)
- **bounds** = a view's location and size using its **own coordinate system** (important for placing the view's content or subviews within itself)

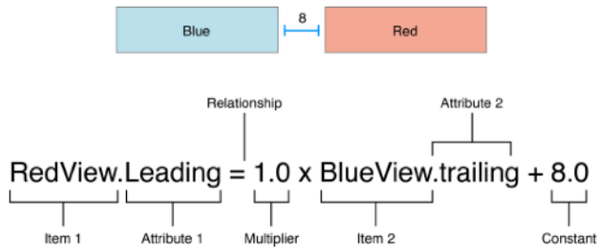
- IMPORTANT:
  - https://medium.com/@studymongolian/frame-vs-bounds-in-ios-107990ad53ee
  - https://stackoverflow.com/questions/5361369/uiview-frame-bounds-and-center

- IMPORTANT:
  - If a view's transform property does not contain the identity transform, the frame of that view is undefined and so are the results of its autoresizing behaviours.
  - So if you do need to move a view around in the parent after a transformation has been done, you can do it by changing the view.center coordinates. Like frame, center uses the coordinate system of the parent view.

02

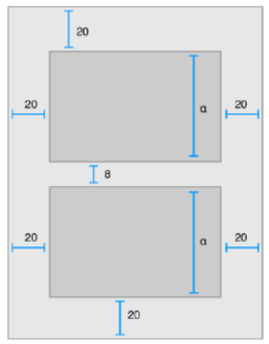
Autolayout

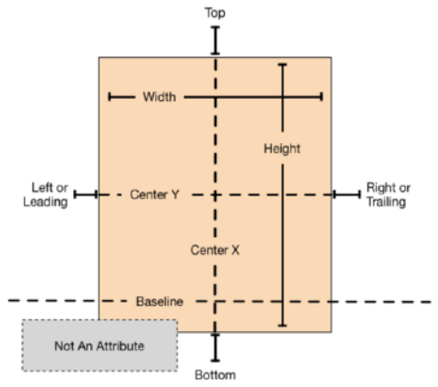
Autolayout - Linear equations



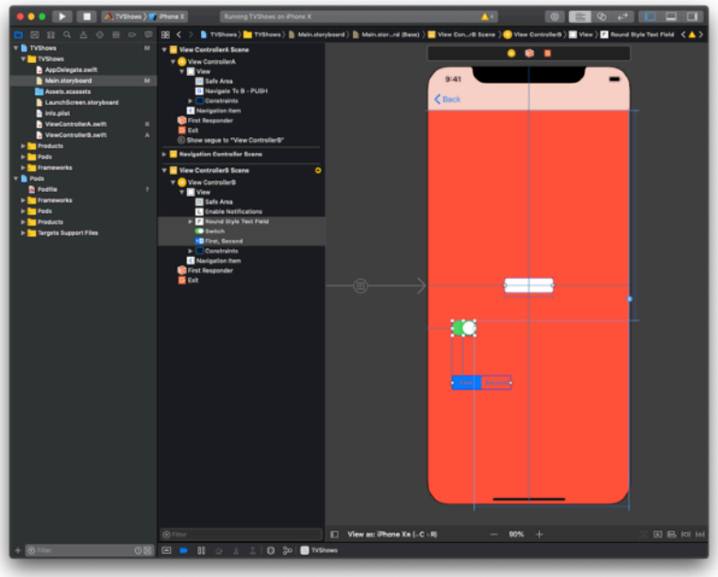
- You want to define your views using simple linear equations that has only one possible solution
- In layman terms:
    - position red view 8 points from the right side of blue one
- Other possibilities
    - or positions red view in the centre of its parent view
    - or positions blue view at 1/3 of its parent height
- https://sudonull.com/post/1599-Mathematical-Fundamentals-Auto-Layout

Autolayout - Linear equations





- How this actually looks on a screen with multiple views
- On the left you have some view and its constraints
- On the right you have possible attributes (stuff that you can use to constrain the view)

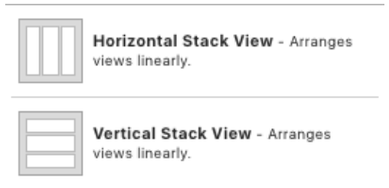


- Here you can see various UI components and their respective constraints
- On the left from the UI canvas, you can see all of the UI components that you use in a specific view and their order of appearance
    - by order I mean from bottom to top, or better yet from back to front in respect to the viewer
- Intrinsic height - “default” height so we dont need (or cannot) specify the height

03

Stack Views

UIStackView

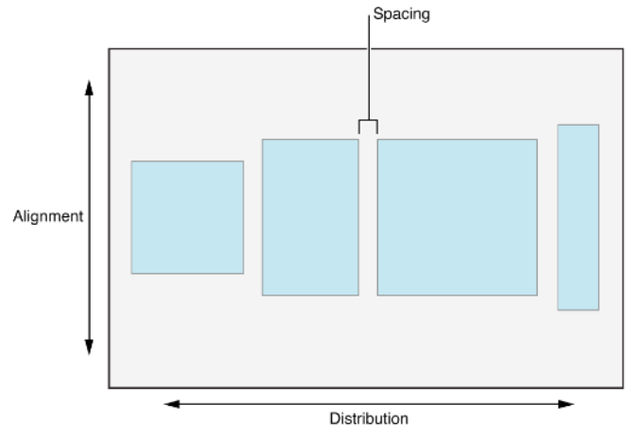


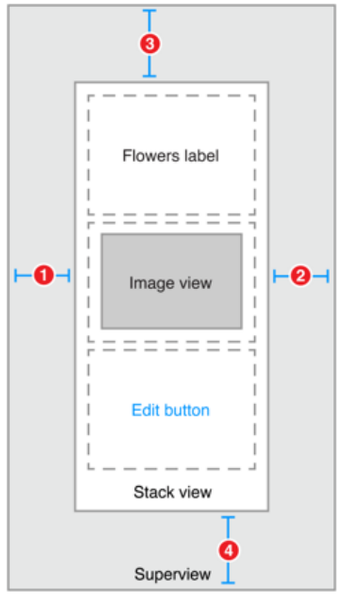
neverstop

- IMPORTANT:
    - Apple preferred way of building UI, meaning in most cases you start with stack view

- Easy way to build your UI
- Can be either vertical or horizontal
- Nesting allowed
- Streamlined way of building UI without the need for so many AutoLayout constraints
- You can mix and match with AutoLayout

UIStackView - Attributes





- Just so that you get mental image of stack views ;)
- Here we have a Label, image in button
- We only set 4 constraints, everything else is solved automatically

04

How to use AutoLayout

Using AutoLayout

- Through **Interface Builder**
  - Probably the easiest and most practical way
- **Programmatically**
  - Somewhat harder
- **Mixed**, using outlets
  - You can set up an **outlet** for a constraint
  - Dynamically updating layout (spacing, sizes etc.)
  - Useful for animations
- **Note**: setting frames and using AutoLayout in most cases won't play nice together

- The most important thing here is to know that when changing frame manually, e.g. changing `view.frame.size.height` will not trigger the AutoLayout update.

NSLayoutConstraint API

```
NSLayoutConstraint(
    item: myView,
    attribute: .trailing,
    relatedBy: .equal,
    toItem: view,
    attribute: .trailingMargin,
    multiplier: 1.0,
    constant: 0.0
).isActive = true
```


- NSLayoutConstraint API
- A bit ugly, not so easy to read

NSLayoutAnchor API

```
NSLayoutConstraint.activate([
    showHUDButton
        .centerXAnchor
        .constraint(equalTo: view.centerXAnchor),
    showHUDButton
        .centerYAnchor
        .constraint(
            equalTo: view.centerYAnchor,
            constant: 64
        )
])
```


- NSLayoutAnchor API
- Much nicer to use and read
- Apples new API
- Before that, we usually used some third party DSL
    - SnapKit
    - Masonry
    - …

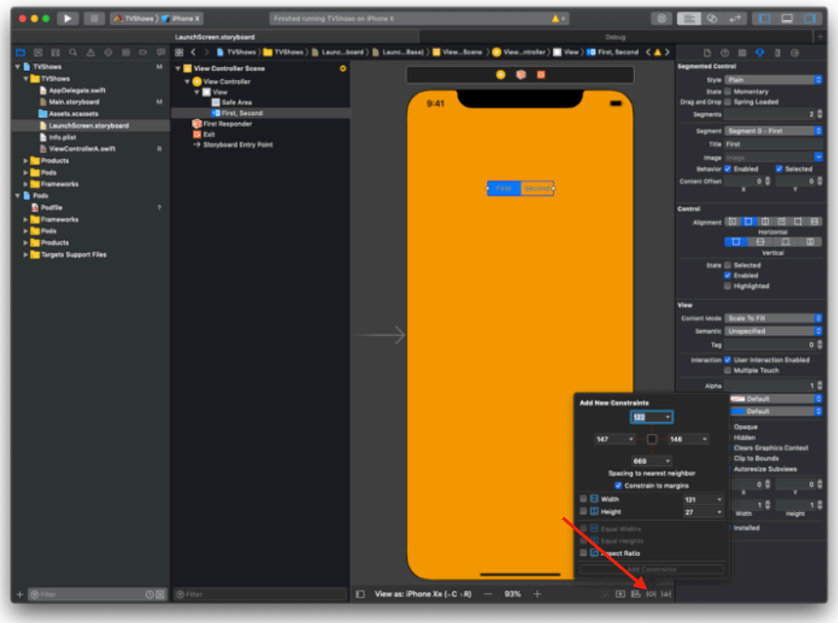
SnapKit
Alternative SDK

```
myView.snp.makeConstraints { make in
    make.center.equalToSuperview()
    make.height.equalTo(150)
    make.leading.trailing.equalToSuperview().inset(16)
}
```


- An alternative wrapper to Apples solution
- Uses the NSLayoutAnchor API in the back
-

Select your view

Click on **Add New Constraint** icon

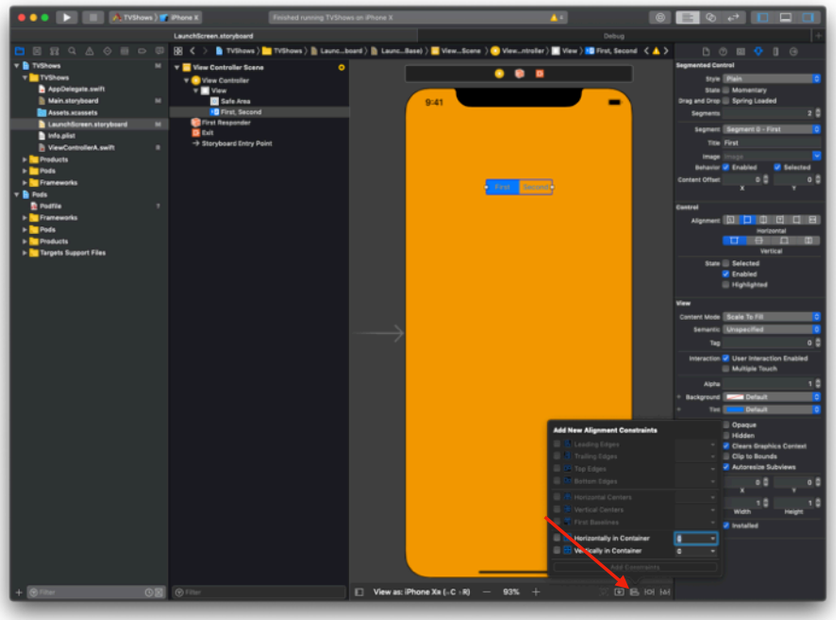


neverstop

- We will use this trough out the course, and also any time we can

Select your view

Click on **Add New Alignment Constraint** icon

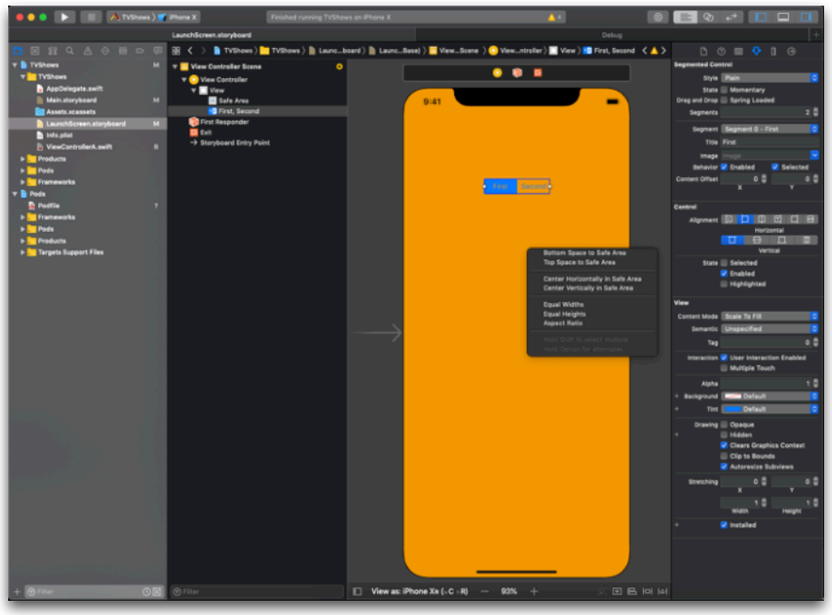


neverstop

- We will use this trough out the course, and also any time we can

Select your view

CRTL + Left Mouse Click



neverstop

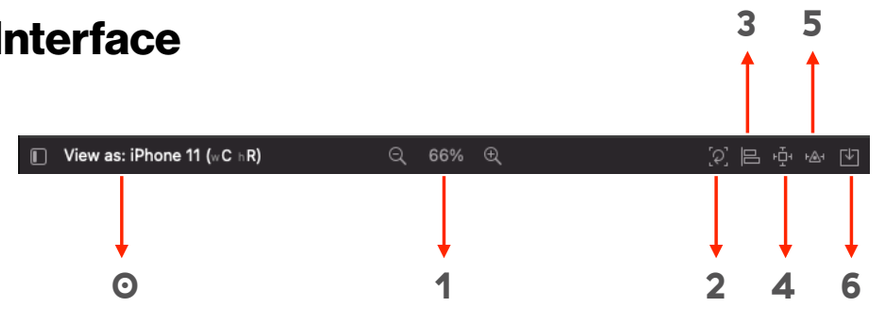
- CTRL + Left Mouse click and drag to other UI element on the screen

Anchoring Elements

- In order for an element to be **properly anchored** using AutoLayout, it needs to have the following properties defined **clearly and unambiguously:**
  - X position
  - Y position
  - Width
  - Height
- They can either be **fixed**, or **relative**, but need to be defined by a **constraint**
- **Intrinsic content size** (UILabel, UIButton...)

- Some UI elements have something called `**intrinsic content size**`
- If we use system UIButton, his intrinsic content size will be based on the text that is inside, or an image.
- So in most cases, you will not need to specify height and width for UIButton.
- Same goes for UILabel

Xcode Interface



0 - View sizes
1 - Zoom in/out
2 - Refresh constraints/view
3 - Align
4 - Pin
5 - Resolve auto layout issues***
6 - Wrap selected components in stack view

*** in most cases won't do what you want to achieve

neverstop

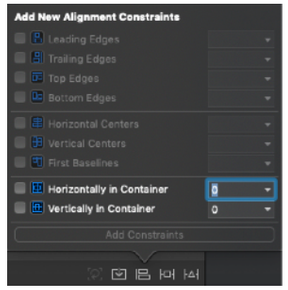
- 0 - Choose different form factors, pretty useful
- 1 - Zoom in and out, very nice when you have small components
- 2 - Refresh view when you update constraints
- 4 - Align multiple components
- 5 - Pin tool, choose where you want to add constraint
- 5 - Resolve auto-layout issues
  - I don’t suggest using auto resolve button ;)
  - in most cases won't do what you want to achieve
- 6 - Select components you want to wrap in stack view and punch it

Pin tool

- Used to **anchor** an element relative to it's **neighbours** or relative to it's **superview**
- Can be used to provide rules for alignment of multiple elements
- In most cases - **turn off constraint to margins**

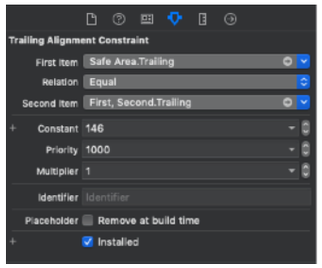
Align tool

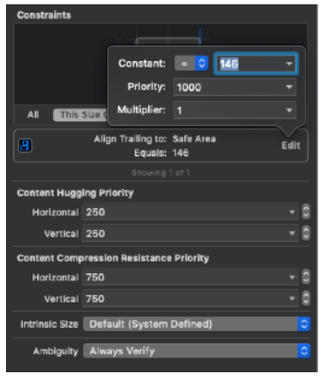
- Used to give rules to **align multiple elements** or to **align a view** based on it's superview
- If the options are **greyed out**, you need to **select multiple elements** in the IB before using them



Examining and editing constraints

- Use the **right sidebar**
- Editing a constraint here will **update the frame** immediately
- Can add **additional rules**

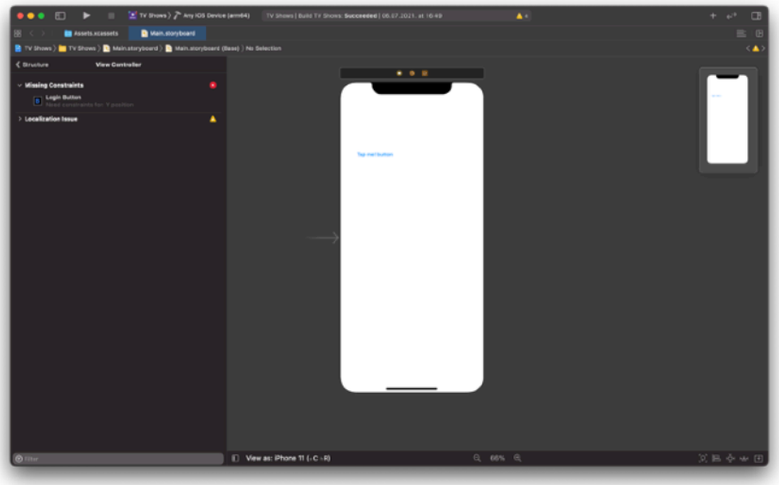




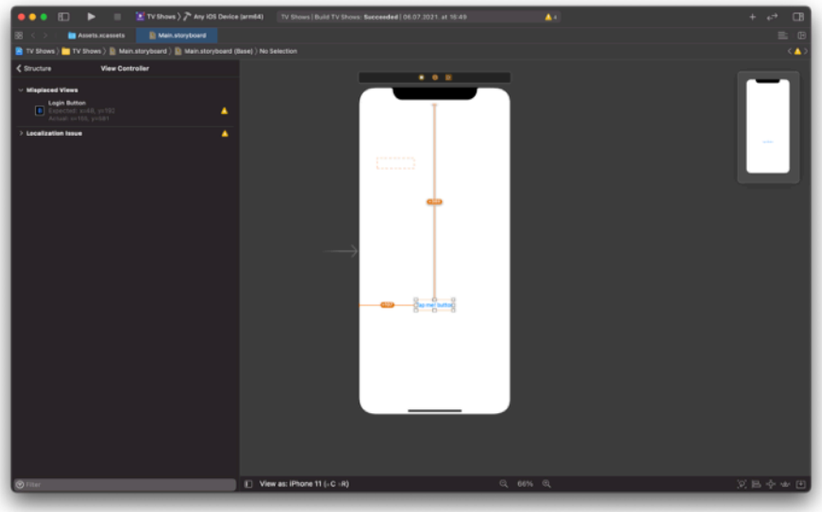
05

How to use Debug

Resolving issues: Missing Constraint



Resolving issues: Misplaced View



Demo time