Auto Layout

Approaches to Interface Layouts

There are basically four approaches to laying out a user interface in iOS:

- **Frame Based Layout**: Setting a view's frame programmatically. In some ways, there is a lot more power and control with this approach; however, it's typically a lot more time-intensive.

- **Autoresizing Masks**: A way to partially automate how a view responds to external change. (We won't cover this)

- **Swift UI**: A new reactive framework introduced in **iOS 13** (We won't cover this)

- **Auto Layout**: A robust system to express constraints between views so they know how to adapt to change

Auto Layout

- Dynamically calculates the **size and position** of all the views in a hierarchy, based on **constraints** placed on those views

- This **constraint-based approach** allows your app to respond when the **size** or **shape** of your **superview** changes. Some common changes:

    1. The device rotates

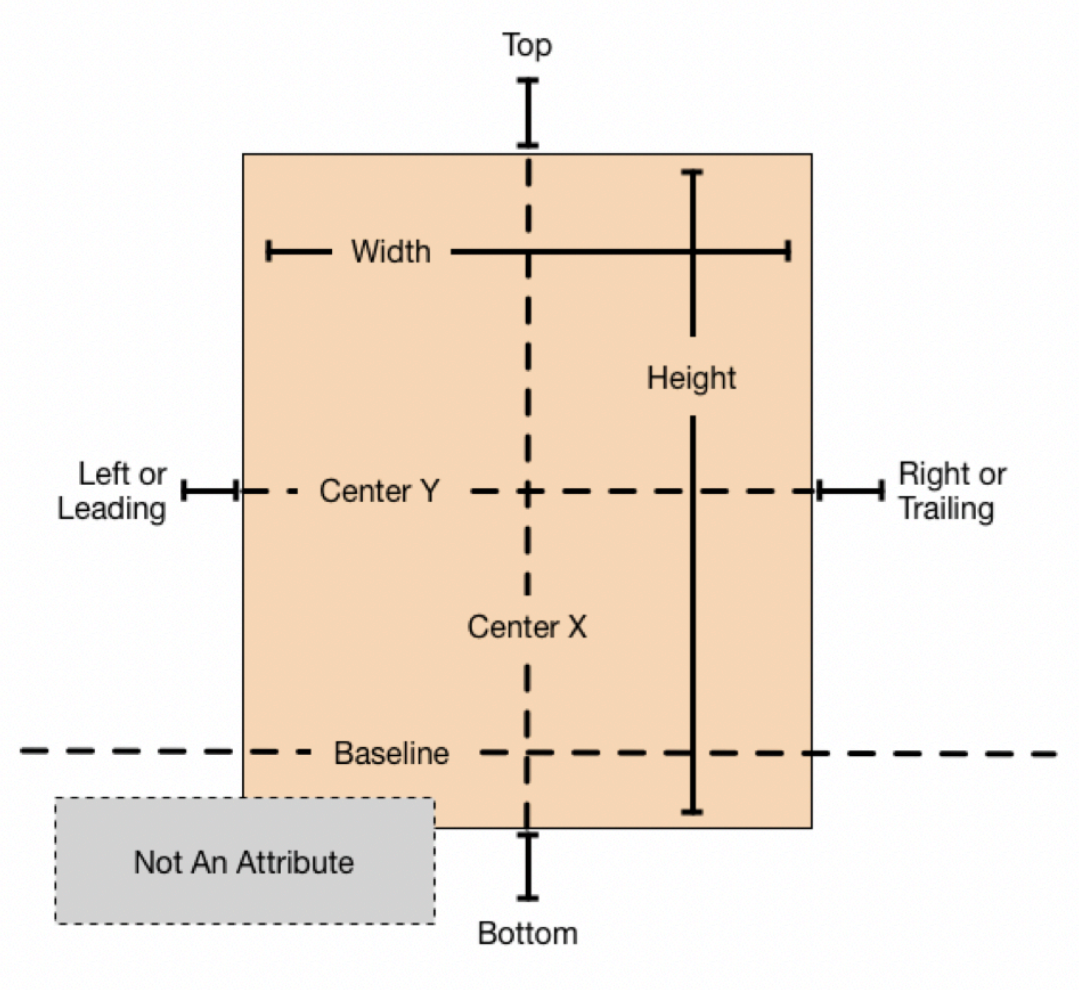
    2. The active call and audio recording bars appear/disappear

    3. You want to support different screen sizes / size classes

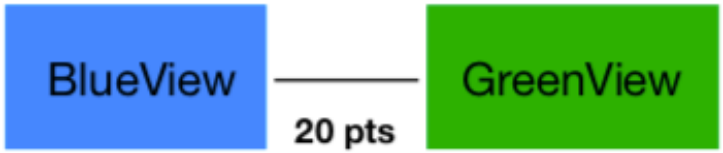
    4. The user enters or leaves Split Screen on an iPad that supports it

Constraint Terminology

- Top & Bottom
- Leading & Trailing (Left and Right for most systems)
- Width & Height
- Center Y & Center X
- Baseline
- Safe Area
- Superview



Anatomy of a Constraint



We want a relationship between these two attributes:

- GreenView.leading = BlueView.trailing

Here's the real formula we need to use:

- GreenView.leading = (1.0 * BlueView.trailing) + 20

It's equivalent to this:

- BlueView.trailing = (1.0 * GreenView.leading) - 20

Why?

Intrinsic Content Size

UIViews need to able to calculate the view's **origin** and **size** in order to correctly position it on the screen.

Some UI components have an **intrinsic size** that is dynamic and based on **current content**.

- Example: **UILabels** define **both** the **height** and the **width** based on the current content. So you only **need** to give it constraints on the origin. (You can still constrain its height and/or width if you like)

Intrinsic Content Size Examples

UIView	No intrinsic content size.
Labels, buttons, switches, and text fields	Height and width.
Sliders	Only width.
Image Views	None when empty. As soon as you add an image, it is set to image size
Text Views, Table Views, and other views that inherit from UIScrollView	In the vast majority of cases, you will need/want to treat these as if they have no intrinsic content size

UIStackView

A streamlined component for laying out a **collection of arranged subviews** in either a **column** or a **row.**

- **Axis** - Vertical or horizontal
- **Distribution** - How the arranged subviews are distributed along the stack view's axis
- **Alignment** - Alignment of arranged subviews perpendicular to the stack view's axis
- **Spacing** - Distance in points between the adjacent edges of the arranged subviews
- **Intrinsic Size** - Based on the size of its arranged subviews

UIStackView