Ambiguous Layouts

Ambiguous layouts occur when the system of constraints has two or more valid solutions. There are two main causes:

- The layout needs additional constraints to uniquely specify the position and location of every view.

  After you determine which views are ambiguous, just add constraints to uniquely specify both the view’s position and its size.

- The layout has conflicting optional constraints with the same priority, and the system does not know which constraint it should break.

  Here, you need to tell the system which constraint it should break, by changing the priorities so that they are no longer equal. The system breaks the constraint having the lowest priority first.

Detecting Ambiguous Layouts

As with unsatisfiable layouts, Interface Builder can often detect, and offer suggestions to fix, ambiguous layouts at design time. These ambiguities appear as warnings in the issues navigator, errors in the document outline, and red lines in the canvas. For more information, see Identifying Unsatisfiable Constraints.

As with unsatisfiable layouts, Interface Builder cannot detect all possible ambiguities. Many errors can be found only through testing.

When an ambiguous layout occurs at runtime, Auto Layout chooses one of the possible solutions to use. This means the layout may or may not appear as you expect. Furthermore, there are no warnings written to the console, and there is no way to set a breakpoint for ambiguous layouts.

As a result, ambiguous layouts are often harder to detect and identify than unsatisfiable layouts. Even if the ambiguity does have an obvious, visible effect, it can be hard to determine whether the error is due to ambiguity or to an error in your layout logic.

Fortunately, there are a few methods you can call to help identify ambiguous layouts. All of these methods should be used only for debugging. Set a breakpoint somewhere where you can access the view hierarchy, and then call one of the following methods from the console:

- hasAmbiguousLayout. Available for both iOS and OS X. Call this method on a misplaced view. It returns YES if the view's frame is ambiguous. Otherwise, it returns NO.
- exerciseAmbiguityInLayout. Available for both iOS and OS X. Call this method on a view with ambiguou
- constraintsAffectingLayoutForAxis:. Available for iOS. Call this method on a view. It returns an array of all the constraints affecting that view along the specified axis.
- constraintsAffectingLayoutForOrientation:. Available for OS X. Call this method on a view. It returns an array of all the constraints affecting that view along the specified orientation.
- _autolayoutTrace. Available as a private method in iOS. Call this method on a view. It returns a string with diagnostic information about the entire view hierarchy containing that view. Ambiguous views are labeled, and so are views that have translatesAutoresizingMaskIntoConstraints set to YES.

You may need to use Objective-C syntax when running these commands in the console. For example, after the breakpoint halts execution, type `call [self.myView exerciseAmbiguityInLayout]`into the console window to call the `exerciseAmbiguityInLayout` method on the `myView` object. Similarly, type `po [self.myView autolayoutTrace]` to print out diagnostic information about the view hierarchy containing `myView`.

NOTE

Be sure to fix any issues found by Interface Builder before running the diagnostic methods listed above. Interface Builder attempts to repair any errors it finds. This means that if it finds an ambiguous layout, it adds constraints so that the layout is no longer ambiguous.

As a result, `hasAmbiguousLayout` returns `NO`. `exerciseAmbiguityInLayout` does not appear to have any effect, and `constraintsAffectingLayoutForAxis:` may return additional, unexpected constraints.