Visual Format Language

This appendix shows how to use the Auto Layout Visual Format Language to specify common constraints, including standard spacing and dimensions, vertical layout, and constraints with different priorities. In addition, this appendix contains a complete language grammar.

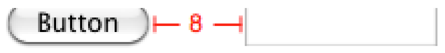
Visual Format Syntax

The following are examples of constraints you can specify using the visual format. Note how the text visually matches the image.

Standard Space

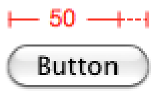
```
[button]-[textField]
```

On This Page



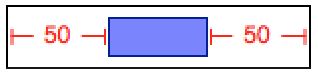
Width Constraint

```
[button(>=50)]
```



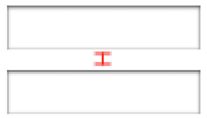
Connection to Superview

```
|-50-[purpleBox]-50-|
```



Vertical Layout

```
V:[topField]-10-[bottomField]
```



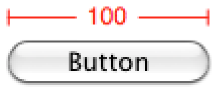
Flush Views

```
[maroonView][blueView]
```

Priority

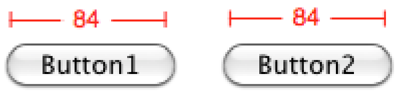
```
[button(100@20)]
```

On This Page



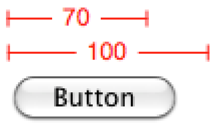
Equal Widths

```
[button1(==button2)]
```



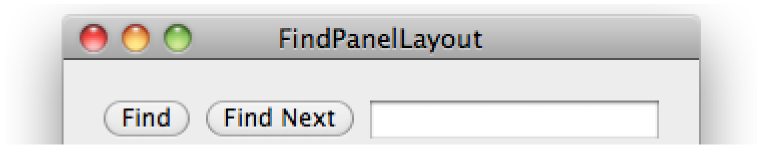
Multiple Predicates

```
[flexibleButton(>=70,<=100)]
```



A Complete Line of Layout

```
|-[find]-[findNext]-[findField(>=20)]-|
```



The notation prefers good visualization over completeness of expressibility. Most of the constraints that are useful in real user interfaces can be expressed using visual format syntax, but there are a few that cannot. One useful constraint that cannot be expressed is a fixed aspect ratio (for example, `imageView.width = 2 * imageView.height`). To create such a constraint, you must use `constraintWithItem:attribute:relatedBy:toItem:attribute:multiplier:constant:`.

Visual For

The visual format string grammar is defined as follows (literals are shown in `code font`; **e** denotes the empty string).

Symbol	Replacement rule
<visualFormatString>	(<orientation>:)? (<superview><connection>)? <view>(<connection><view>)* (<connection><superview>)?
<orientation>	H|V
<superview>	|
<view>	[<viewName>(<predicateListWithParens>)?]
<connection>	**e**|–<predicateList>–|–
<predicateList>	<simplePredicate>|<predicateListWithParens>
<simplePredicate>	<metricName>|<positiveNumber>

Symbol	Replacement rule
<predicateListWithParens>	(<predicate>(,<predicate>)*)
<predicate>	(<relation>)?(<objectOfPredicate>)(@<priority>)?
<relation>	==|<=|>=
<objectOfPredicate>	<constant>|<viewName> *(see note)*
<priority>	<metricName>|<number>
<constant>	<metricName>|<number>
<viewName>	Parsed as a C identifier. This must be a key mapping to an instance of `NSView` in the passed views dictionary.
<metricName>	Parsed as a C identifier. This must be a key mapping to an instance of `NSNumber` in the passed metrics dictionary.
<number>	As parsed by `strtod_l`, with the C locale.

On This Page

> NOTE
>
> For the `objectOfPredicate` production, `viewName` is acceptable only if the subject of the predicate is the width or height of a view. That is, you can use `[view1(==view2)]` to specify that `view1` and `view2` have the same width.

If you make a syntactic mistake, an exception is thrown with a diagnostic message. For example:

```
Expected ':' after 'V' to specify vertical arrangement
V|[backgroundBox]|
 ^

A predicate on a view's thickness must end with ')' and the view must end with ']'
|[whiteBox1][blackBox4(blackWidth][redBox]|
                                 ^

Unable to find view with name blackBox
```

```
|[whiteBox2][blackBox]
                     ^

Unknown relation. Must be ==, >=, or <=
V:|[blackBox4(>30)]|
               ^
```

On This Page