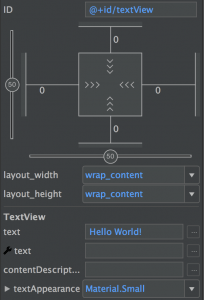
ConstraintLayout – Часть 1

[3 Июня, 2016](https://blog.stylingandroid.com/constraintlayout-part-1/)[Марк Эллисон](https://blog.stylingandroid.com/author/admin/)[3 Комментариев](https://blog.stylingandroid.com/constraintlayout-part-1/#comments)

В Google ИО 2016 Google анонсировал новый Android-макета имени *ConstraintLayout*. Несмотря на первоначальный выпуск быть помечены как Альфа-релизе, это на самом деле очень стабильный (с некоторыми оговорками). В этой серии статей мы взглянем на этот новый макет-малыш на блоке и попытаться сделать некоторые выводы, как лучше использовать его.

[](https://blog.stylingandroid.com/constraintlayout-part-1/constraint-properties/)Прежде чем мы начнем, стоит делать некоторые отличия:*ConstraintLayout* полностью отделен от нового редактора. Новый макет редактор не предназначены для работы с *ConstraintLayout*он также хорошо работает с другими типами макета. Кроме того, новый редактор макета на самом деле больше, чем просто редактор – это также позволяет визуальное редактирование ресурсов меню, а также раскладок, так я вообще буду ссылаться на это как визуальный редактор, а не редактор. Были предыдущие попытки, обеспечивающим визуальное редактирование макета, который, в то время как благими намерениями, были несколько не хватает в их реализации – они не дружат с пользовательский*вид*ы, например. Именно эти уроки, извлеченные из этих предыдущих попыток визуального макета редакторов, что новый редактор был переписан с нуля – и это видно – это уже гораздо более способен обрабатывать пользовательский *вид*с при условии, что вы держите конструкторов довольно чистым и в полной мере использовать View#isInEditMode() , чтобы убедиться, что он не пытается сделать слишком много, когда он используется в визуальном редакторе.

Так что, хотя *ConstraintLayout* отличается от редактора, два не идти, а рука-в-руку и*ConstraintLayout* был разработан, чтобы быть обработаны посредством визуального редактора, а не ручной работы с XML. Я говорил пару Гуглеров, которые бы играли с *ConstraintLayout* и оба заявили, что они даже не считают ручной разработке XML-файле просто потому, что визуальный редактор был настолько хорош. Некоторые первоначальные опасения были озвучены разработчиками о том, что на самом деле быть в состоянии видеть XML, но опасения оказались недолговечны, как только вы начнете играть с ним, потому что визуальный редактор поддерживает установленный образец внизу вкладки, позволяющие переключаться между текстом (XML) и дизайн (визуальный редактор) вид. Что сказал, *ConstraintLayout* XML-это довольно объемный и не самый простой, чтобы понять как результат.

Обсудив *ConstraintLayout* с рядом разработчиков Android на ИО было общее ощущение, что*ConstraintLayout* был в целом хорошо, но некоторые люди выражают озабоченность по поводу того *ConstraintLayout* будет приятно играть с *CoordinatorLayout* и если анимация *ConstraintLayout*будет сложно. Обе эти опасения имеют под собой основания:

Первым делом стоит отметить о *ConstraintLayout* заключается в том, что (хотя мы пока не имеем источников для *ConstraintLayout*, вы можете увидеть декомпиляции если развернутьclasses.jar на constraint-layout-1.0.0-alpha2 внешней библиотеки в Android Studio проекта посмотреть) он расширяет *ViewGroup*. Это довольно важно, потому что так делают все более традиционной компоновки, которые очень хорошо играют с *CoordinatorLayout*. Кроме того, если вы на самом деле создать новый основной деятельности, используя Новый Android-деятельность мастер (Right click module|New...|Activity|Basic Activity), то мастер автоматически создаст свой *ConstraintLayout* для основного содержимого внутри*CoordinatorLayout* и можно назначить поведение на *ConstraintLayout* именно так, как вы можете с другими планировками.

Так что насчет анимации *ConstraintLayout*? В очередной раз это просто *ViewGroup* , так что он (и его дети) могут быть анимированы в точности так же, как и любой другой *ViewGroup*. Если этого не будет сразу ответить на поставленный вам вопрос, то давайте сделаем шаг назад к макетам 101 и напоминаем о том, как макеты и *посмотреть*на самом деле работать. Макет-это просто*ViewGroup* , который отвечает за позиционирование своего ребенка *красивый*с. Прежде чем мы можем нарисовать макет и своих детей мы в первую очередь необходимо выполнить макет пропуска для расчета и настройки положения и размеров ребенка *смотреть*с. Как правило, это будет включать первоначальную оценку сдать, чтобы определить Внутренние размеры ребенка*смотреть*, и затем вычислить положение и размер каждого ребенка перед тем как установитьmLeft, mTop, mRight, и mBottom значений для каждого ребенка относительно родителя. После передачи макета была завершена, то каждого ребенка *вид*имеет свои границы, установленные внутри, поэтому, когда мы приходим, чтобы нарисовать макет, просто вызвав *функцию ondraw()*на каждого ребенка будет обращено на правильное положение и размер.

Когда мы подошли к оживляющие вещи, мы не хотим вносить изменения в сам макет, так как запуск передачи макета для каждого кадра анимации будет слишком дорого и убивают наших сменой кадров. Вместо этого мы выполняем схема прохода перед установкой анимации и просто анимации позиций личности ребенка *вид*ы, изменив их translationX, translationYиtranslationZ атрибутами (звоню setX() на *вид на* самом деле изменить его translationXзначение, а не его mLeft стоимость, например). Так это на самом деле не важно, какой родительского макета конкретный *вид* и, когда мы пришли, чтобы оживить это все, что имеет значение заключается в том, что родительский макет рассчитала ее mLeft, mTop, mRight, иmBottom значения уже.

Так что теперь, если мы рассматриваем *CoordinatorLayout* снова, это должно быть даже понятнее, что любой анимации, которые запускаются в результате *CoordinatorLayout* поведения реализаций полностью зависит от того, используем ли мы *ConstraintLayout* по сравнению с любой другой макет.

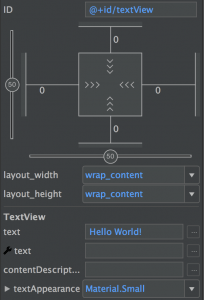
Надеюсь, это теперь должно быть ясно, что *ConstraintLayout* будет вести себя не иначе, чем другие макеты, когда речь идет о выполнении анимации – он просто работает заранее любой анимации, а при настройке анимации фактический родитель макет, который отвечает за позиционирование и изменение размеров конкретного ребенка *вид* совсем не имеет значения.

Так что с некоторые мифы и опасения развеяла, в следующей статье мы начнем искать в*ConstraintLayout* и посмотреть, как это работает.

Часть 2

[10 Июня 2016](https://blog.stylingandroid.com/constraintlayout-part-2/)[Марк Эллисон](https://blog.stylingandroid.com/author/admin/)[3 Комментариев](https://blog.stylingandroid.com/constraintlayout-part-2/#comments)

В Google ИО 2016 Google анонсировал новый Android-макета имени *ConstraintLayout*. Несмотря на первоначальный выпуск быть помечены как Альфа-релизе, это на самом деле очень стабильный (с некоторыми оговорками). В этой серии статей мы взглянем на этот новый макет-малыш на блоке и попытаться сделать некоторые выводы, как лучше использовать его.

[](https://blog.stylingandroid.com/constraintlayout-part-1/constraint-properties/)Прежде чем попасть в вшивый песчаный из *ConstraintLayout*следует краткое обсуждение о том, почему это необходимо. Первая причина-это производительность. Это была хорошо документирована, что глубокие иерархии макета стоят очень дорого в плане измерения и разметка проходит. Мы также сказали, что гнездиться взвешенным *LinearLayouts* является неэффективным, и его следует избегать – это потому, что у каждого ребенка должна быть измерена дважды и так, как мы их гнездо, то количество измерений каждого ребенка растет в геометрической прогрессии на каждый вложенный планировка уровня. Аналогично [*уровень мельче* и похож двойного измерения](https://medium.com/google-developers/developing-for-android-iii-2efc140167fd#.glttq3kre) вопросы и советы приходят из Google, чтобы избежать их как контейнер верхнего уровня.

Вложенные, утяжеленным *макетом* вопрос был частично рассмотрен *PercentLayout*, и *уровень мельче* вопрос был частично решен с помощью *GridLayout*. Однако, как многие разработчики будут знать, иногда надо идти против советов, поступающих от Google просто потому, что эти макеты являются те, которые лучше всего подходит для создания макетов, наши дизайнеры дают нам.

Это также подводит нас к другой вопросы, которые топ-уровня планировки следует использовать? Получение такой выбор права является вековой проблемой в Android развития, и это то, что нельзя научить – он становится почти инстинктивным с опытом. Однако, никакой опыт не может защитить от дизайнера делая некоторые якобы незначительные изменения в макет, которые имеют большие последствия с точки зрения иерархии макета и требует полной реорганизации рабочего макета.

*ConstraintLayout* специально разработан для замены *уровень мельче* и в большинстве случаев требуется только однократное измерение ребенка *смотреть*в схема прохода – в некоторых случаях двойных измерений не требуется, но разработчики в курсе этих и либо устранить их или, по крайней мере, уменьшить влияние на производительность. Он также предназначен, чтобы быть по умолчанию верхнего уровня контейнера макета и лучшие практики, чтобы избежать вложенности другие макеты внутри *ConstraintLayout*. Удерживая макет уплощена таким образом, он также снижает экспоненциальный рост Размеры для случаев двойного измерения, потому что мы не должны добавить макетов ребенка.

Основная концепция *ConstraintLayout* заключается в том, что ребенка *смотреть*с есть ряд опорных точек для которых ограничения могут быть прикреплены к левой, верхней, правой и нижней кромок; или базовой линии текста для виджетов, которые содержат текст. Ограничение одной анкерной точке от одного *вида* к якорю точки братика *посмотреть*, или родитель. Эти ограничения обеспечивают подобные функции на *уровень мельче.Ни* атрибутами. Аналогичным образом на *уровень мельче* мы можем задать поля для создания смещения. Но это еще не все – помимо этого мы также можем установить значения смещения, которые позволяют нам создать смещений на основе соотношений пространства. Эти смещения значений, используемых в сочетании с различными layout\_[width|height] значениями могут дать нам хороший контроль над тем, как отдельного ребенка *вид*ы на самом деле размера.

Эти понятия все немного абстрактно и трудно на самом деле понять, как они работают по отдельности, а уж в сочетании друг с другом. Однако причина, начиная с этого высокого уровня объяснение отчасти определить некоторые названия, которые мы будем полагаться на позже, и отчасти для того, чтобы посадить семя, что большая часть мощности *ConstraintLayout* это то, каким образом мы можем добиться казалось бы, сложные макеты только объединив некоторые из этих принципов атомной.

Кроме того, одним из первых критику *ConstraintLayout* заключается в том, что это будет трудно рассмотреть код, потому что разработчики не будут знакомы с форматом XML. Цель этой серии, чтобы сломать некоторые понятия в этих более атомов компонентов для того, чтобы сделать его легче читать и понимать XML при рассмотрении запросов на github, оба или другого источника РЕПО, не снимая код на локальном компьютере, чтобы просмотреть его в визуальном редакторе. Конечно, мы всегда можем начать в том числе и скриншотов из визуального редактора в рамках нашего описания пр, чтобы сделать его проще для наших коллег-разработчиков, чтобы понять наши планы, но, будучи в состоянии понять XML будет бонус, а также.

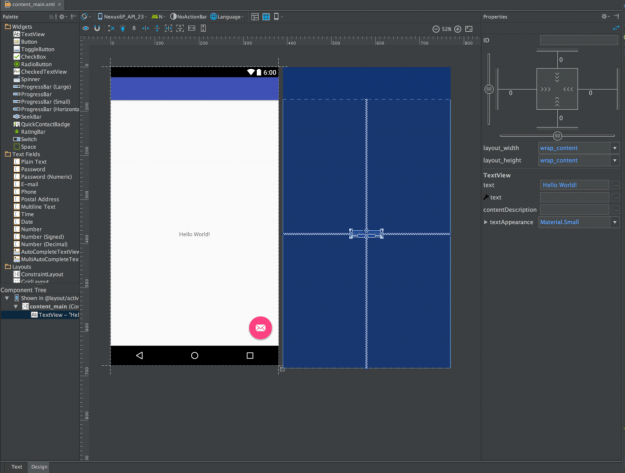
Следующим шагом является на самом деле использовать визуальный редактор, чтобы создать макет и посмотреть, как мы можем достичь различных вещей, совместив эти понятия, но также взглянуть на результирующем XML, чтобы начать понимать макет просто по прочтению XML-файлов. В следующей статье мы будем делать именно это.

Часть 3

[17 Июня 2016](https://blog.stylingandroid.com/constraintlayout-part-3/)[Марк Эллисон](https://blog.stylingandroid.com/author/admin/)[1 Комментарий](https://blog.stylingandroid.com/constraintlayout-part-3/#comments)

В Google ИО 2016 Google анонсировал новый Android-макета имени *ConstraintLayout*. Несмотря на первоначальный выпуск быть помечены как Альфа-релизе, это на самом деле очень стабильный (с некоторыми оговорками). В этой серии статей мы взглянем на этот новый макет-малыш на блоке и попытаться сделать некоторые выводы, как лучше использовать его.

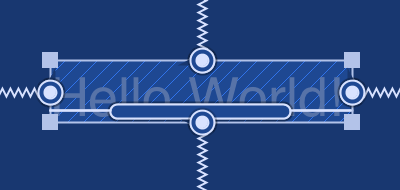
Теперь, когда мы ознакомились с основными понятиями о *ConstraintLayout* давайте обратим наше внимание на визуальный редактор, чтобы посмотреть, как мы можем создавать ограничения. Если мы начнем с создания нового проекта с основной деятельностью студии в Android 2.2 предварительного просмотра 1 или поздно он создаст нам два макета activity\_mainи content\_main. Это стандартный шаблон для шаблон основной деятельности. Мы также получаем меню генерируется автоматически, и это может использоваться, чтобы проверить визуальный редактор меню. Мы фокусируемся на *ConstraintLayout* поэтому мы заинтересованы в content\_main, так что если мы откроем и перейдите в режим конструктора (используя вкладки внизу):

[](https://blog.stylingandroid.com/?attachment_id=3907)

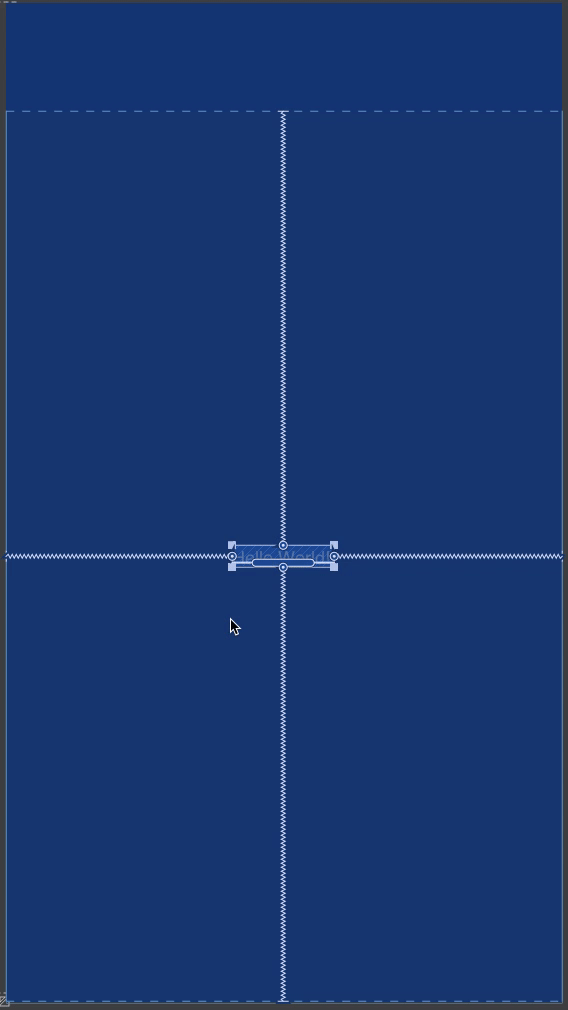
На левой стороне у нас есть палитра, которая содержит доступных виджетов, которые можно перетаскивать в наш макет, и ниже, что является компонентом дерево, которое показывает иерархию наш макет и все составные виджеты. В середине у нас есть главная панель редактор, который включает в себя как предварительный просмотр WYSISYG нашего макета, и новое представление копирку. Справа находится панель свойств, на которой отображаются свойства выбранного виджета.

Как мы фокусируемся на *ConstraintLayout*, я не собираюсь идти во все подробности нового визуального редактора, и наши главные направления будут главный редактор и свойства панелей. Однако компонент дерева может быть очень полезным в нескольких случаях. Во-первых, если вы боретесь, чтобы выбрать определенный виджет на главной панели редактора, потому что у него есть и другие виджеты перед ним, вы можете выбрать его в дереве компонентов. Вторая очень полезная вещь заключается в том, что вы можете перетаскивать виджеты изменять их порядок в макете, и поэтому порядок, в котором они нарисованы в.

Как я уже говорил, подавляющее большинство нашего внимания при создании макетов с*ConstraintLayout* корень собирается быть главным редактором панель и панель свойств. Просмотр чертежа показывает все установленные в настоящее время ограничения в макете, а в графическом представлении отображаются только их, когда он имеет фокус (с мышкой внутри его границ). Когда вы выберите виджет в виде его количество ручек появится вокруг него:

[](https://blog.stylingandroid.com/?attachment_id=3910)

Квадратные ручки на углах могут быть использованы для изменения размера виджета; круговой на каждой кромке, ограничение привязки точек; и удлиненной округло-прямоугольник в центре-это текст базовой точки привязки. На изображении мы видим, что у нас есть ограничения, добавленные в каждой из пограничных якоря (в зигзагообразные линии представляют собой особый вид ограничений – об этом чуть позже). При наведении указателя мыши на один из этих якорей, которые есть ограничение добавленные щелкнув по ней снимут ограничения – обратите внимание, как виджет textview, переходит к верхней части экрана, если как снять нижнее ограничение, потому что все, что остается-это главное ограничение:

[](https://blog.stylingandroid.com/?attachment_id=3914)

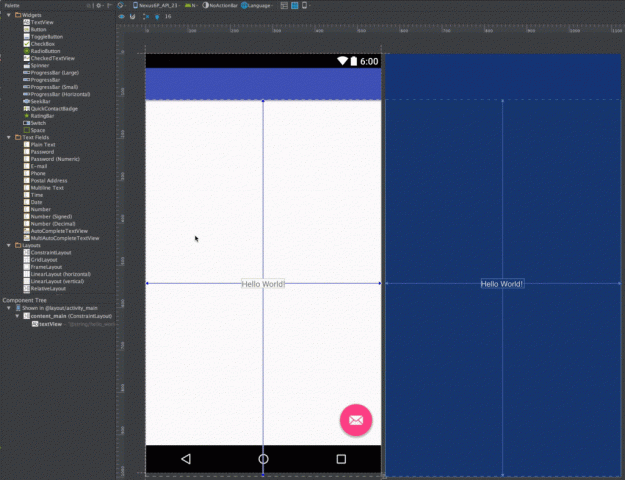
Мы можем создать новое ограничение, нажав на опорную точку, а затем перетащить к точке привязки на брата, или в данном случае *ConstraintLayout* себя – обратите внимание, как перед тем, как отпустить кнопку мыши в поле textview переходит к позиции, которую он будет занимать одно ограничение было создано рассказав о том, как это будет выглядеть:  
[](https://blog.stylingandroid.com/?attachment_id=3919)

Правила создания ограничений на самом деле очень проста: Вы можете создать только ограничение от горизонтальной опорной точке (слева или справа; или базовой точки привязки) на горизонтальную опорную точку на другой виджет или родитель *ConstraintLayout*; и вы можете только создать ограничение от вертикали опорную точку (сверху или снизу) на вертикальную опорную точку на другой виджет, или родитель *ConstraintLayout*.

Если мы перетащите еще один *элемент textview* ниже существующей, обратите внимание, как он встанет на полях, которые мы наметили в 16dp с помощью переключателя на панели инструментов:

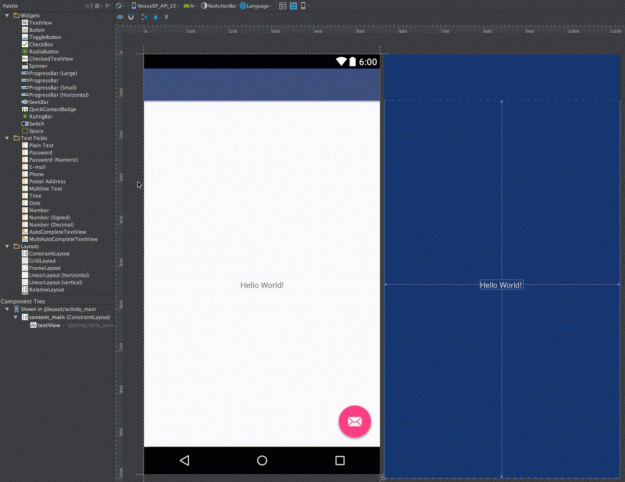
[панель инструментов (маржинальной)](https://blog.stylingandroid.com/?attachment_id=3921)

Когда мы перетащить его в грубой установки мы можем создавать ограничения для установки его относительно существующего *виджет textview*:

[](https://i2.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/05/Add-widget-without-autoconnect.gif)  
Существует альтернативный механизм можно использовать, хотя. Если мы включить автосоединение тумблер на панели инструментов:

[панель инструментов (автосоединение)](https://blog.stylingandroid.com/?attachment_id=3925)

На этот раз, если мы перетащите его примерно в положение автоматически создает некоторые ограничения по умолчанию на основе, где дите уронил:

[](https://i0.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/Add-widget-with-autoconnect.gif)Изначально очень медленной анимации появляются немного режет слух, потому что они, кажется, слишком долго. Однако причина этого в том, что он дает вам пару секунд, чтобы передумать после того, как вы сбросили новый виджет. Вы не должны ждать его, чтобы закончить, если вы счастливы, вы можете просто продолжать работать и просто игнорировать его.

Итак, давайте взглянем на XML, что это на самом деле производит:

res/layout/content\_main.xml



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | <?Формат XML Версия="1.0" кодирование="utf-8"не?>  <андроид.поддержка.ограничения.ConstraintLayout описание:андроид="http://schemas.android.com/apk/res/android"    описание:приложение="http://schemas.android.com/apk/res-auto"    описание:инструментов="http://schemas.android.com/tools"    андроид:идентификатор="@+ID и/content\_main"    для Android:layout\_width="присваиваем"    ОС Android:layout\_height="присваиваем"    приложение:layout\_behavior="@строка/appbar\_scrolling\_view\_behavior"    инструменты:контекст="ком.stylingandroid.constraintlayout.В mainactivity"    инструменты:покажу="@макета/activity\_main">      <Виджет textview      андроид:идентификатор="@+ID или виджет textview"      ОС Android:layout\_width="wrap\_content в качестве параметров"      ОС Android:layout\_height="wrap\_content в качестве параметров"      андроид:layout\_marginBottom="16dp"      для Android:гравитация="центр"      андроид:текст="@строка/она"      приложение:layout\_constraintBottom\_toBottomof="@+ID и/content\_main"      приложение:layout\_constraintLeft\_toLeftOf="@+ID и/content\_main"      приложение:layout\_constraintRight\_toRightOf="@+ID и/content\_main"      приложение:layout\_constraintTop\_toTopOf="@+ID и/content\_main" />      <Виджет textview      андроид:идентификатор="@+по ID/textView1"      ОС Android:layout\_width="wrap\_content в качестве параметров"      ОС Android:layout\_height="wrap\_content в качестве параметров"      андроид:layout\_marginTop="16dp"      андроид:текст="@строка/поле textview"      приложение:layout\_constraintLeft\_toLeftOf="@+ID или виджет textview"      приложение:layout\_constraintRight\_toRightOf="@+ID или виджет textview"      приложение:layout\_constraintTop\_toBottomOf="@+ID и/виджет textview" />    </андроид.поддержка.ограничения.ConstraintLayout> |

Итак, мы видим, что родительская *ConstraintLayout* имеет layout\_behaviour применены которое управляет поведения в его родительской CoordinatorLayout (лишнее доказательство того, что они играют хорошо вместе). Глядя на первый *элемент textview,* мы видим ограничения, которые соответствуют ограничениям мы видим в визуальном редакторе что центр ее внутри родительского элемента. Глядя на второе *поле textview* мы можем увидеть подобные ограничения, которые соответствуют в визуальном редакторе, положение его по отношению к первому *виджет textview*.

Поэтому создание ограничений достаточно проста, и XML, которые они производят на самом деле довольно проста и понятна.

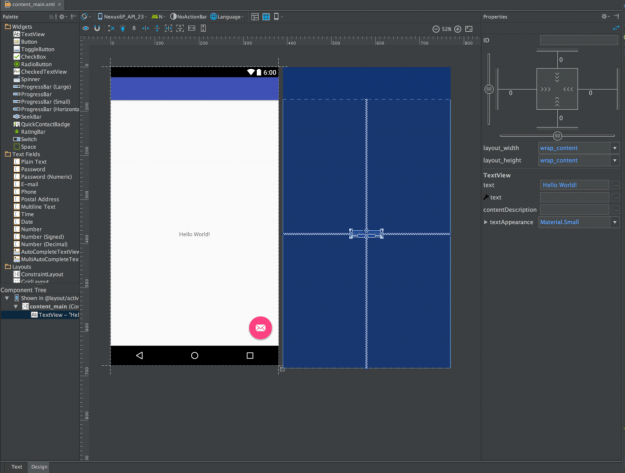
В следующей статье мы рассмотрим, как изменить некоторые стандартные атрибуты макет, который мы уже знакомы с может измениться, как эти ограничения будут применены к виджета.

Часть 4

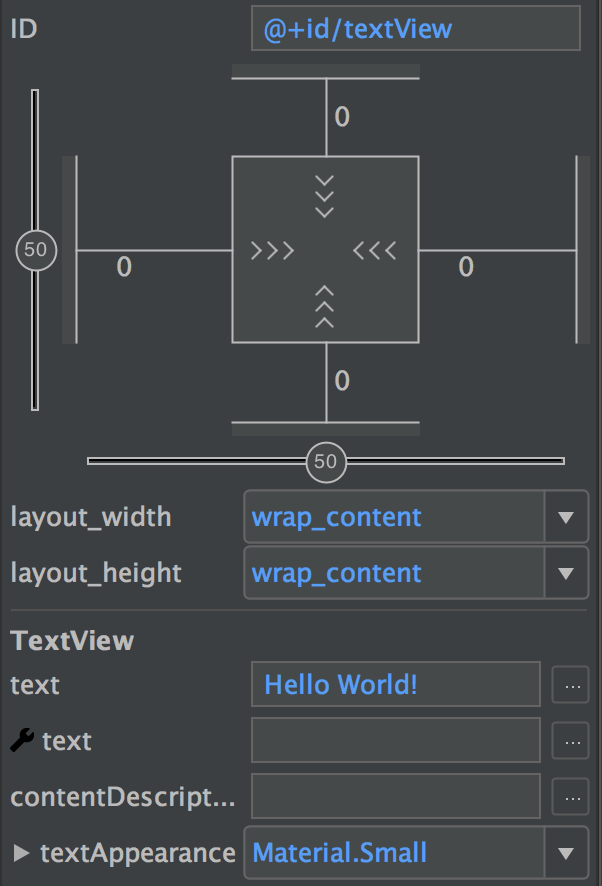
[24 июня, 2016](https://blog.stylingandroid.com/constraintlayout-part-4/)[Марк Эллисон](https://blog.stylingandroid.com/author/admin/)[без комментариев](https://blog.stylingandroid.com/constraintlayout-part-4/#respond)

В Google ИО 2016 Google анонсировал новый Android-макета имени *ConstraintLayout*. Несмотря на первоначальный выпуск быть помечены как Альфа-релизе, это на самом деле очень стабильный (с некоторыми оговорками). В этой серии статей мы взглянем на этот новый макет-малыш на блоке и попытаться сделать некоторые выводы, как лучше использовать его.

В прошлой статье мы рассмотрели, как создавать ограничения для установки виджета относительно другой или родитель *ConstraintLayout*, но мы также можем настроить размер и положение с помощью нескольких компоновочных параметров на дочерние виджеты. На самом деле мы на самом деле обнаруживаем, что мы уже знакомы с некоторыми из этих параметров макета, мы просто должны понять, как они работают в контексте *Constraintlayout*. Давайте посмотрим на визуальный редактор:

[](https://i1.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/05/Editor-Full.png)

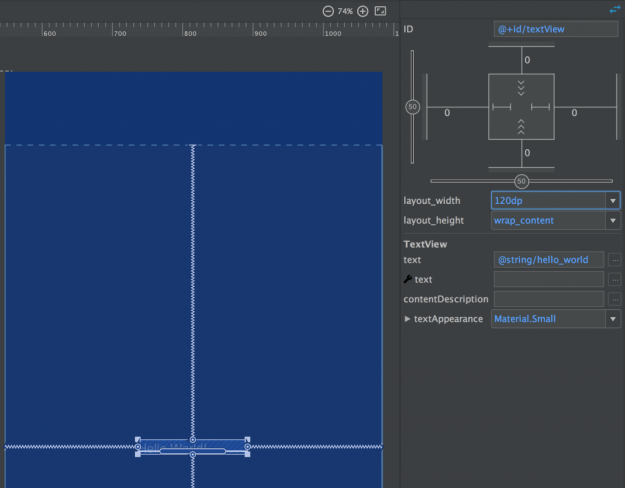
В верхней части имущих панель-это панель, которая контролирует эти:

[](https://i0.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/Constraint-properties.png)

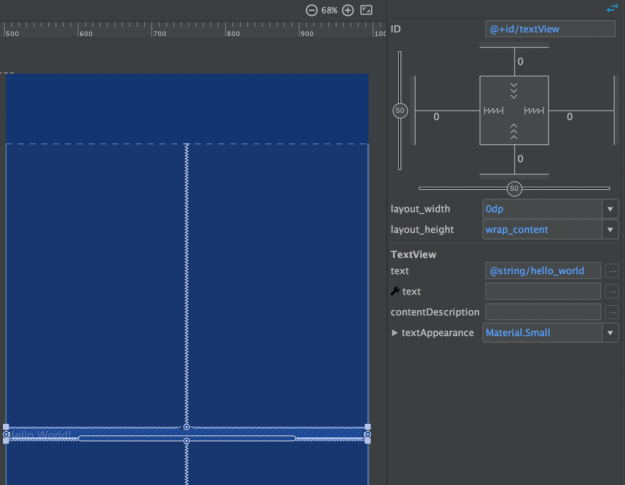
Мы можем видеть некоторые поля, которые должны принять непосредственное чувство к кому-либо с какой-то Android верстка опыт работы: есть вид код вверху, а затем внизу странная схема у нас есть layout\_width и layout\_height атрибуты, а затем некоторые поля в *поле textview*. Это странная схема, что нас интересует, и это не так странно и страшно, как кажется на первый взгляд.  
При наведении указателя мыши на любой из элементов управления в этой схеме подсказке появится, чтобы дать вам намек на его функции.

[символ layout_width](https://i0.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/layout_width-symbol.png)Давайте начнем в середине и выход. В центре схемы мы имеем эти символы-стрелки:

Это контролирует внутренние Размер *виджет textview,* который в настоящее время выбран, или поставить его в условия, которые должны быть знакомы ему соответствуетlayout\_width="wrap\_content". Да, действительно, это так просто, что стрелки просто означать,wrap\_content что, собственно говоря, отражено в *layout\_width* поле на панели свойств. Однако, есть также две другие возможные пути, которые мы можем заполнять layout\_width атрибут и мы можем переключаться между тремя государствами, просто нажав на этот символ. Следующее состояние является фиксированной ширины – значение по умолчанию будет измеренный Размер *смотреть*, но вы можете изменить его (в идеале вы должны использовать это значение дименс, но я использую сырое значение ДП для ясности):

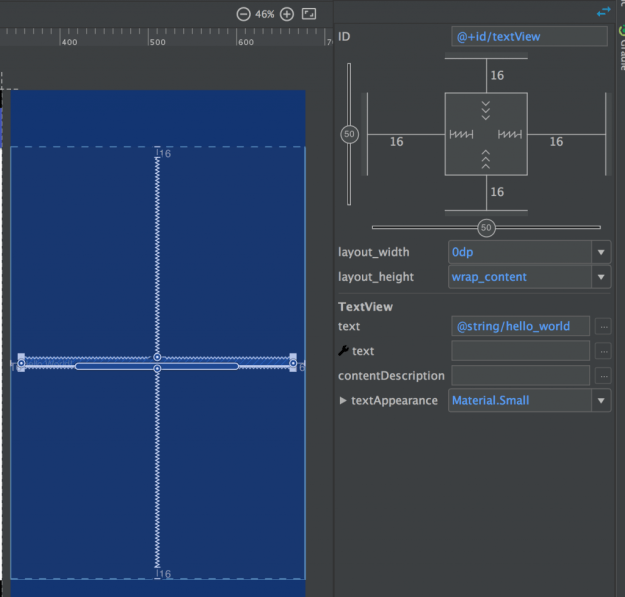
[](https://i2.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/layout_width-fixed.png)

Третье состояние называется “любой размер” и означает, что вид будет Размер, чтобы соответствовать доступное пространство. Это смутно, по аналогии с match\_parent (который не поддерживается *ConstraintLayout* – если вы укажите это действительно можно сделать любого размера поведение). Любой размер поведением фактически ближе к объединениюlayout\_width="0dp" и layout\_weight="1" при использовании взвешенного *элемента linearlayout*. В нашем простом примере это будет вести себя в точности, как match\_parent но, как мы увидим позже в этой серии, любой размер может не соответствовать родительским Размер, следовательно, решение не поддерживать match\_parent , потому что имя на самом деле сообщают в *ConstraintLayout* контексте:

[](https://i1.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/layout_width-Any-Size.png)

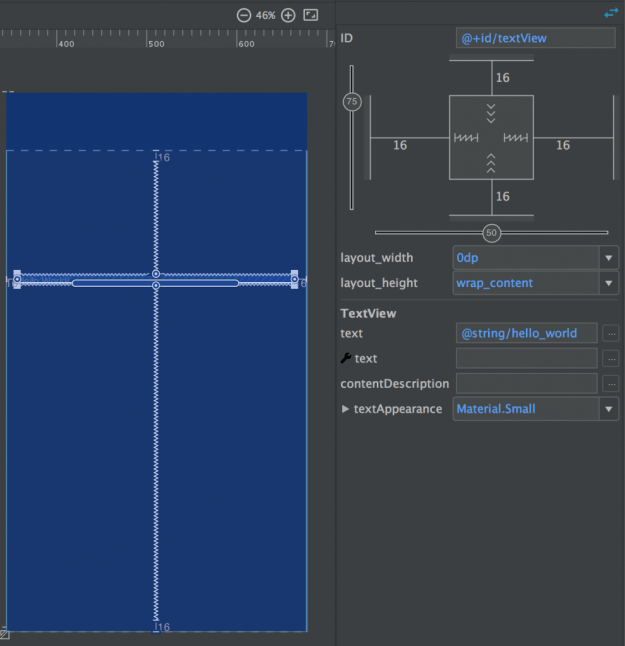
Конечно вертикальные символы ведут себя точно таким же образом, за исключением их контроля layout\_height.

Так что если мы идем, мы видим несколько строк с числами рядом с ними, и если мы нажмем на один из номеров мы Вам выпадающем списке, содержащих числа: 0, 8, 16, 24, & 32.Это набор индивидуальных layout\_margin\* атрибутов виджета. Еще раз повторюсь, это просто графический редактор для атрибутов, которые мы уже знакомы, и, если мы установить поля16dp все вокруг (Вы не должны определить ДП – это будет на самом деле привести к [краху](https://code.google.com/p/android/issues/detail?id=212166" \t "_blank)Студия Android 2.2 просмотр 2 Если вы делаете) мы видим, как это влияет на позиционирование горизонтальных ребер целью с любым размером поведения:

[](https://i1.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/margins.png)

Обратите внимание, как Ширина *виджет textview* теперь смещение от родительского объекта,16dp – и это видно в проекте панели.

Переезжаю в очередной раз на схему, мы приходим к “предвзятости” настройки – это первые новые типы параметров, которые мы видели. Если посмотреть на ранние экрана крышки вы можете видеть, как *поле textview* сосредоточена в его родителе *ConstraintLayout*. Это потому, что у нас есть два ограничения, применяемые к *виджет textview* в каждом направлении. Например, в вертикальном направлении у нас есть ограничение сверху на *элемент textview,* который привязан к родительским *ConstraintLayout*, и один снизу на *элемент textview,* который привязан к нижней части материнской *ConstraintLayout*. Потому что его внутренняя высота имеет значение *wrap\_content в качестве параметров*, в *поле textview* сосредоточена в родительском, потому что две вертикальные ограничения применяются в равной степени. Однако, если мы поменяем “вертикального смещения” можно вообще применить два ограничения в различных соотношениях и разместите *виджет textview* при различной относительной позиции в работе с родителями:

[](https://i0.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/vertical-bias.png)

Еще один способ думать об этом является то, что ограничения на самом деле являются две пружины якорь мнение ребенка в своего родителя – они представлены в панели копирку. Если пружины такие же сила, то мнение ребенка будет по центру, но если мы изменим силу пружины можно расположить различные относительные положения в родительском – и поэтому уровень защиты делает.

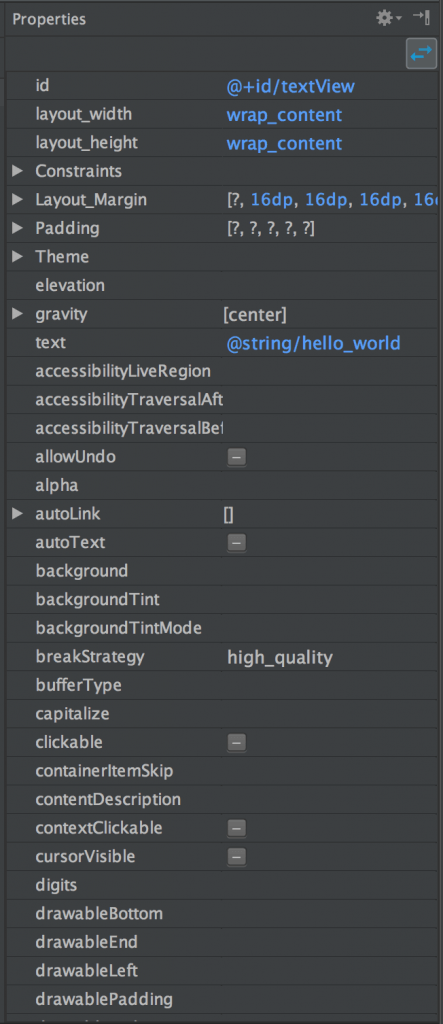
res/layout/content\_main.xml



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | <андроид.поддержка.ограничения.ConstraintLayout описание:андроид="http://schemas.android.com/apk/res/android"    описание:приложение="http://schemas.android.com/apk/res-auto"    описание:инструментов="http://schemas.android.com/tools"    андроид:идентификатор="@+ID и/content\_main"    для Android:layout\_width="присваиваем"    ОС Android:layout\_height="присваиваем"    приложение:layout\_behavior="@строка/appbar\_scrolling\_view\_behavior"    инструменты:контекст="ком.stylingandroid.constraintlayout.В mainactivity"    инструменты:покажу="@макета/activity\_main">      <Виджет textview      андроид:идентификатор="@+ID или виджет textview"      ОС Android:layout\_width="wrap\_content в качестве параметров"      ОС Android:layout\_height="wrap\_content в качестве параметров"      андроид:layout\_marginBottom="16dp"      андроид:layout\_marginEnd="16dp"      андроид:layout\_marginStart="16dp"      андроид:layout\_marginTop="16dp"      для Android:гравитация="центр"      андроид:текст="@строка/она"      приложение:layout\_constraintBottom\_toBottomof="@+ID и/content\_main"      приложение:layout\_constraintHorizontal\_bias="0.20"      приложение:layout\_constraintLeft\_toLeftOf="@+ID и/content\_main"      приложение:layout\_constraintRight\_toRightOf="@+ID и/content\_main"      приложение:layout\_constraintTop\_toTopOf="@+ID и/content\_main"      приложение:layout\_constraintVertical\_bias="0.25" />    </андроид.поддержка.ограничения.ConstraintLayout> |

Итак, мы уже должны понимать, как размер и рентабельности представлены в XML, поэтому я не стану их здесь. Единственной новой вещью являются параметры смещения и представлены несколько новых атрибутов, которые принимают значение между 0.0 и 1.0, чтобы представить позицию относительно родителя,

свойства переключателяТак что это все атрибуты, которые представлены на схеме в панели свойств. Одна вещь, стоит отметить, что хотя в панели свойств не содержит все возможные атрибуты, которые могут быть установлены на определенный вид, нажав на кнопку в правом верхнем углу переключит на другой панели, в котором перечислены все доступные атрибуты:

[](https://i0.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/full-properties.png)

С помощью этой панели мы можем изменить любой из доступных атрибутов для определенного ракурса и не нужно редактировать XML-данные непосредственно, чтобы сделать так.

Хотя теперь у нас есть некоторые инструменты, которые позволяют нам делать очень интересные и доселе сложные вещи в Android макеты, есть еще один чрезвычайно мощный компонент *ConstraintLayout* которую мы рассмотрим в следующей статье.

Часть 5

[1 июля 2016](https://blog.stylingandroid.com/constraintlayout-part-5/)[Марк Эллисон](https://blog.stylingandroid.com/author/admin/)[без комментариев](https://blog.stylingandroid.com/constraintlayout-part-5/#respond)

В Google ИО 2016 Google анонсировал новый Android-макета имени *ConstraintLayout*. Несмотря на первоначальный выпуск быть помечены как Альфа-релизе, это на самом деле очень стабильный (с некоторыми оговорками). В этой серии статей мы взглянем на этот новый макет-малыш на блоке и попытаться сделать некоторые выводы, как лучше использовать его.

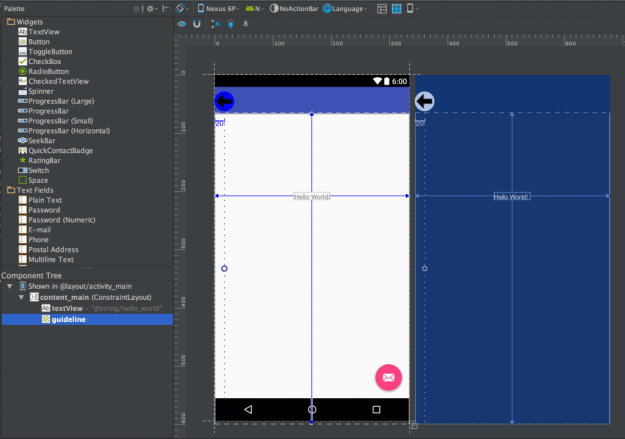
До сих пор мы изучили, как создать ограничения на *вид*, и как мы можем контролировать то, как эти ограничения влияют на размер и расположение, что *посмотреть* в родительском*ConstraintLayout* уже сейчас можно сделать некоторые действительно мощные и интересные вещи с этими инструментами, но что-то может быть очевидно: достижения аналогичного поведения к весовым *макетом* трудно достичь с тем, что мы видели до сих пор. На первый взгляд установка смещении предлагает определенный уровень контроля над *положением* в*целях* но это не позволит нам контролировать Размер. Пока у нас есть любой размер поведения, которые мы можем применить к нашему *мнению*мы все равно не можем контролировать часть пространства, которое каждый *смотреть* будет, как мы можем с утяжеленным *макетом*. Однако есть еще одна особенность, что *ConstraintLayout* , что позволяет нам реализовать такое поведение, хотя и совершенно иным путем: *руководство*с.

Для тех, кто знаком с графическими инструментами редактирования, такие как Adobe иллюстратор, дизайнер сродство, эскиз, фотошоп, и даже некоторые страницы макета и инструментов для обработки текста понятие принципы должны быть достаточно знакомы. Эти несколько строк, которые можно установить на холст правка-время, которое мы можем использовать, чтобы выровнять положение и другие графические элементы, но не будет выводиться на окончательной или на страницу изображения. ConstraintLayout имеет собственную реализацию и получается, что они очень полезны.

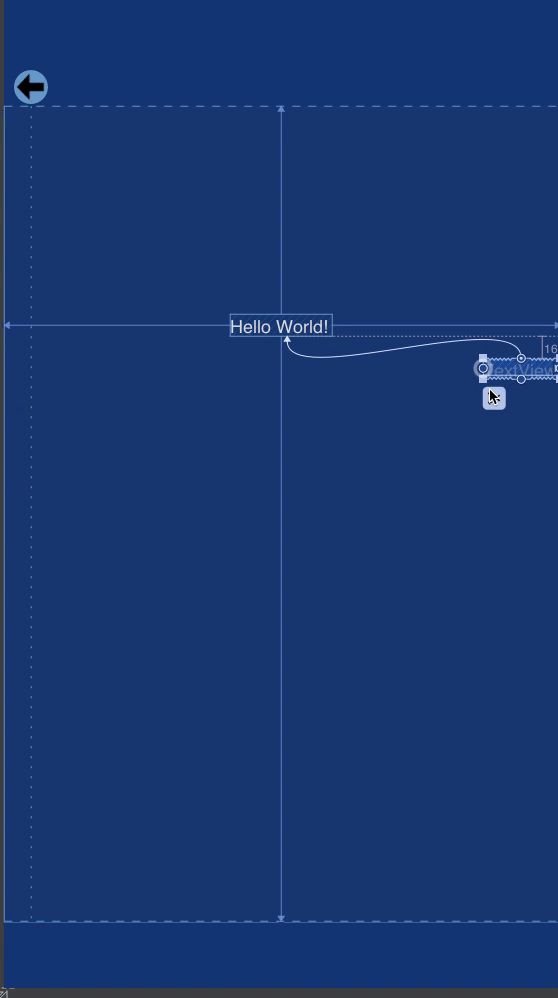
Проблема с *руководящими принципами* в ранних версий было то, что они были не так-то просто найти в новом визуальном редакторе, но от Студия Android 2.2 превью 3 года они были повышены на главной панели инструментов в визуальном редакторе:

[](https://i2.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/07/add-guidelines.png)

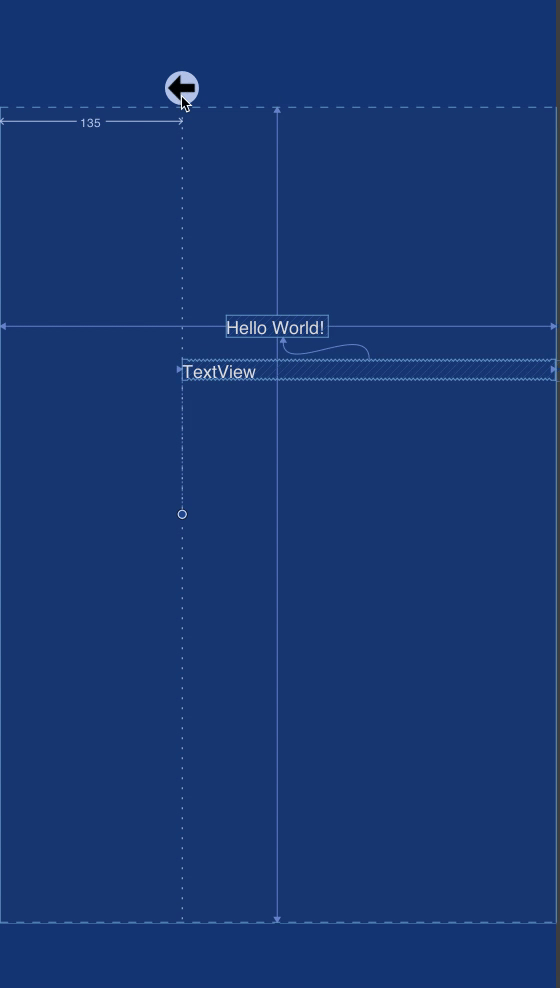
Если добавить вертикальную *направляющую* , мы можем теперь увидеть его в предварительный просмотр и чертеж панели, а также она появляется в *Вид* иерархии в дереве компонентов панели, а также:

[](https://i1.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/vertical-guideline-cropped.png)

Теперь мы можем создавать ограничения относительно *основного положения*, и если мы затем передвиньте *направляющую* после того, как один из более ограничения были присоединены к ней, прилагаемый *посмотреть*будет двигаться вместе с ним:

[](https://i0.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/constraint-to-guideline.gif)

Это все хорошо, но мы движемся *основного положения* мы видим, что его значение положение меняется и это просто смещение от левого края в ДП. Однако существует три различных способа, что мы можем расположить *руководство* – как смещение от левой (или начать) края родителя в ДП; как смещение от правого (или конец) края родителя в ДП; или в относительных процентах от размера материнского (я фокусируюсь на вертикальную *направляющую*для теперь, но то же самое относится к горизонтальным – только верхний и нижний края). Мы можем переключаться между этими режимами позиционирования, нажав на контрольную точку для*ориентира*:

[](https://i2.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/guideline-position-types.gif)

Оно должно быть довольно очевидно, что мы можем разделить родитель *ConstraintLayout*используя *руководящее положение*S и согласования индивидуальных мнений в любом размере поведение *руководящего*S и вам очень сходное поведение (и контроль за их поведением) к весовым *макетом*. Кроме того, мы можем использовать *Руководство*по горизонтальном и вертикальном направлениях для создания табличных данных с одной плоской иерархии без каких-либо вложенных макетов. Это довольно мощная штука.

Так как *руководство* действительно работает? Ныряя в contraint-layout-1.0.0-alpha2 АПК в Android Studio мы можем на самом деле проверить исходный код для *ориентира* и видим, что это подклассы *вид*, но переопределяет несколько методов, чтобы убедиться, что он имеет нулевую высоту и ширину, фактически никогда не нарисовать что-нибудь, и Исправлена видимость View.GONE. Во время передачи макета он будет позиционироваться как любой *вид* в макет пропуска, и другие *вид*ы (или даже другие *Директивы*, если вы хотите начать все начала-нравится!) можно расположить по отношению к ней. Это просто *мнение,* так это может быть ограничение ссылку как и любой другой *вид*. Но он никогда не будет на самом деле сделать, когда вычерчивается.

Так как они появляются в нашем XML-макета:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | <?Формат XML Версия="1.0" кодирование="utf-8"не?>  <андроид.поддержка.ограничения.ConstraintLayout описание:андроид="http://schemas.android.com/apk/res/android"    описание:приложение="http://schemas.android.com/apk/res-auto"    описание:инструментов="http://schemas.android.com/tools"    андроид:идентификатор="@+ID и/content\_main"    для Android:layout\_width="присваиваем"    ОС Android:layout\_height="присваиваем"    приложение:layout\_behavior="@строка/appbar\_scrolling\_view\_behavior"    инструменты:контекст="ком.stylingandroid.constraintlayout.В mainactivity"    инструменты:покажу="@макета/activity\_main">      <андроид.поддержка.ограничения.Руководство      андроид:идентификатор="@+ID и/guideline1"      ОС Android:layout\_width="wrap\_content в качестве параметров"      ОС Android:layout\_height="wrap\_content в качестве параметров"      для Android:ориентация="вертикаль"      приложение:relativeBegin="135dp"      инструменты:layout\_editor\_absoluteX="135dp"      инструменты:layout\_editor\_absoluteY="0dp хиллингдон" />      <андроид.поддержка.ограничения.Руководство      андроид:идентификатор="@+ID и/guideline2"      ОС Android:layout\_width="wrap\_content в качестве параметров"      ОС Android:layout\_height="wrap\_content в качестве параметров"      для Android:ориентация="вертикаль"      приложение:relativeEnd="90dp"      инструменты:layout\_editor\_absoluteX="321dp"      инструменты:layout\_editor\_absoluteY="0dp хиллингдон" />      <андроид.поддержка.ограничения.Руководство      андроид:идентификатор="@+ID и/guideline3"      ОС Android:layout\_width="wrap\_content в качестве параметров"      ОС Android:layout\_height="wrap\_content в качестве параметров"      для Android:ориентация="вертикаль"      приложение:relativePercent="20"      инструменты:layout\_editor\_absoluteX="86dp"      инструменты:layout\_editor\_absoluteY="0dp хиллингдон" />    </андроид.поддержка.ограничения.ConstraintLayout> |

Единственное важных атрибутов app:relative[Begin|End|Precent] , который представляет собой смещение от начала и конца края, и в процентах от родительского макета – поведение, которое мы уже обсуждали.

Много возможностей из-за *руководства*, но есть и некоторые ограничения, которые мы должны рассмотреть. Мы видели, как мы можем управлять положением и размером другой *вид*, который имеют ограничения, прикрепленный к *направляющей*, но имейте в виду, что анимация*руководство* не будет иметь никакого эффекта. Как мы обсуждали в [части 1 этой серии](https://blog.stylingandroid.com/constraintlayout-part-1/), размера и позиции других *вид*ов рассчитываются во время передачи макета, и если вы потом анимировать *основного положения* это не будет иметь никакого влияния на эти*представления*с, если другой макет проходит. Выполнение макета для каждого кадра анимации-это очень плохая идея, потому что пропуск макет стоит дорого, и ваша частота кадров упадет ниже плинтуса. Никто не любит кривые анимации! Вам потребуется применить анимацию индивидуально к *мнению*, что вы хотите анимировать.

Таким образом мы рассмотрели все основные элементы, как *ConstraintLayout* работает, а в следующей статье в этой серии мы будем принимать более шаблонный подход и посмотреть некоторые приемы и методы, которые мы можем достичь, используя *ConstraintLayout*.

Часть 6

[8 Июля 2016](https://blog.stylingandroid.com/constraintlayout-part-6/)[Марк Эллисон](https://blog.stylingandroid.com/author/admin/)[2 Комментария](https://blog.stylingandroid.com/constraintlayout-part-6/#comments)

В Google ИО 2016 Google анонсировал новый Android-макета имени *ConstraintLayout*. Несмотря на первоначальный выпуск быть помечены как Альфа-релизе, это на самом деле очень стабильный (с некоторыми оговорками). В этой серии статей мы взглянем на этот новый макет-малыш на блоке и попытаться сделать некоторые выводы, как лучше использовать его.

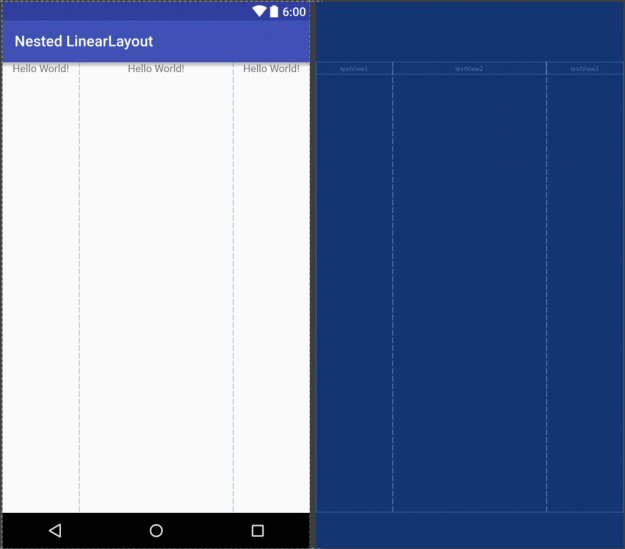
Теперь, когда мы узнали основные принципы *ConstraintLayout* давайте взглянем на то, как применять его на практике. Давайте начнем, где мы затронули в прошлой статье, с имитируя поведение взвешенных краю. Любой, кто следил за серию до сих пор должны иметь хорошее представление о том, как это будет работать, но мы будем в это, тем не менее. Предположим, мы имеем следующую раскладку:

Взвешенный Элемент Linearlayout

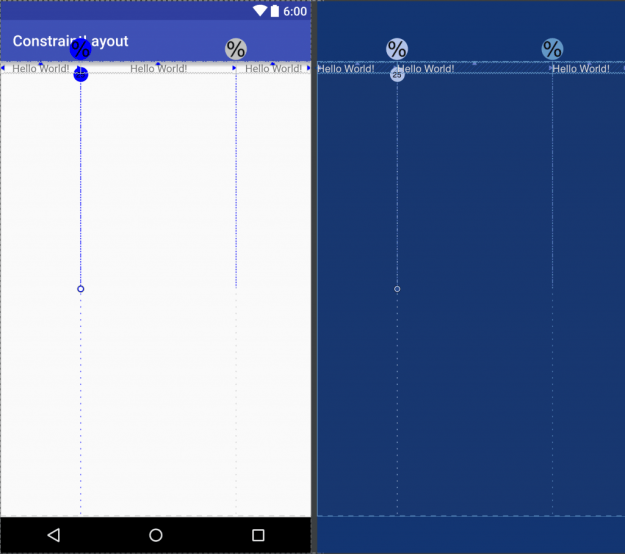


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <?Формат XML Версия="1.0" кодирование="utf-8"не?>  <Макетом описание:андроид="http://schemas.android.com/apk/res/android"    для Android:layout\_width="присваиваем"    ОС Android:layout\_height="присваиваем"    для Android:ориентация="горизонтальный">      <Виджет textview      андроид:идентификатор="@+по ID/textView1"      для Android:layout\_width="0dp хиллингдон"      ОС Android:layout\_height="wrap\_content в качестве параметров"      для Android:layout\_weight="1"      для Android:гравитация="центр"      андроид:текст="@строка/она" />      <Виджет textview      андроид:идентификатор="@+ID и/textView2"      для Android:layout\_width="0dp хиллингдон"      ОС Android:layout\_height="wrap\_content в качестве параметров"      для Android:layout\_weight="2"      для Android:гравитация="центр"      андроид:текст="@строка/она" />      <Виджет textview      андроид:идентификатор="@+ID и/textView3"      для Android:layout\_width="0dp хиллингдон"      ОС Android:layout\_height="wrap\_content в качестве параметров"      для Android:layout\_weight="1"      для Android:гравитация="центр"      андроид:текст="@строка/она" />  </Элемента linearlayout> |

если мы анонсируем это в визуальном редакторе мы видим, как это должно выглядеть:

[](https://i0.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/nested-LinearLayout.png)

Если нам нужно добавить вторую линию нам нужен другой *макет* и мы бы тогда нужно добавить как к горизонтальному *краю* родителя. Однако мы можем сделать то же самое используя *ConstraintLayout* с парой *руководство*с:

[](https://i2.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/06/Nested-Mimic-cropped.png)

Если мы хотим добавить вторую линию для этого, мы просто должны добавить еще три*объекта textview*С и добавить горизонтальные ограничения на *Ориентир*с (точно как в первой строке), и добавить ограничение, чтобы расположить их ниже первой позиции.

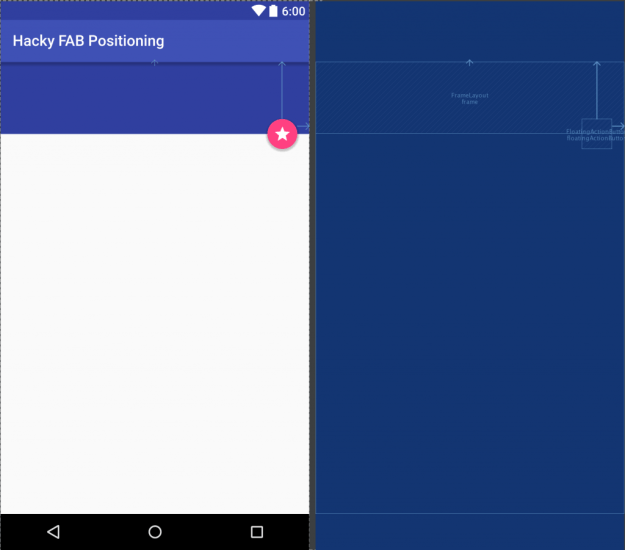
Далее давайте обратим внимание на то, что в реальном мире болевую точку: в материале даются рекомендации по проектированию есть [несколько примеров того, как ФАБ, должен находиться на краю панели](https://www.google.com/design/spec/components/buttons-floating-action-button.html#buttons-floating-action-button-large-screens). Как правило, мы можем сделать это с помощью довольно суховато технику нанесения отступ от верхней части панели. В этом макете @id/frame это панели нужно выровнять, и мы должны применить верхнее поле на ФАБ, чтобы правильно позиционировать его. Значение этого на самом деле зависит от высоты панели и высота ФАБ, и если мы изменим Размер панели, то ФАБ не двигаться вместе с ним. Конечно могут быть и другие способы сделать это, но я просто предоставляя очевидное решение, которое, несомненно, будет использован в много макетов! Макет выглядит так:

Суховато ФАБ позиционирования



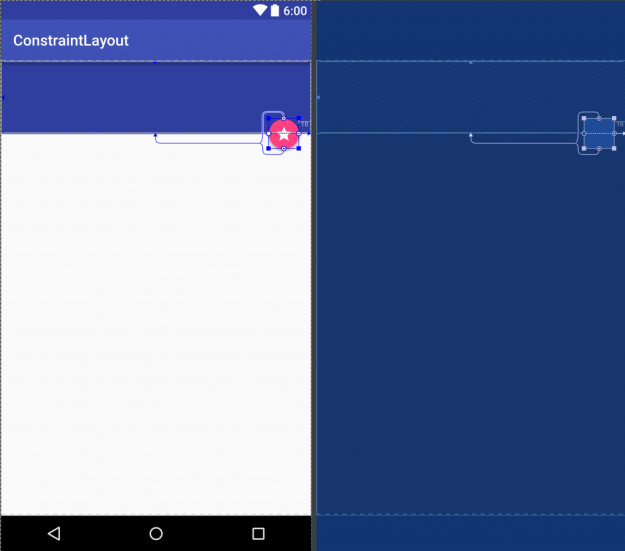
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <?Формат XML Версия="1.0" кодирование="utf-8"не?>  <Уровень мельче префиксом xmlns:андроид="http://schemas.android.com/apk/res/android"    описание:приложение="http://schemas.android.com/apk/res-auto"    для Android:layout\_width="присваиваем"    ОС Android:layout\_height="присваиваем">      <Заменить      андроид:идентификатор="@+ID и/рамка"      для Android:layout\_width="присваиваем"      ОС Android:layout\_height="96dp"      андроид:layout\_alignParentTop="правда"      андроид:фон="@цвет/colorPrimaryDark" />      <андроид.поддержка.дизайн.виджет.FloatingActionButton      андроид:идентификатор="@+ID и/floatingActionButton"      ОС Android:layout\_width="wrap\_content в качестве параметров"      ОС Android:layout\_height="wrap\_content в качестве параметров"      андроид:layout\_alignEnd="@+ID и/рамка"      андроид:layout\_alignTop="@+ID и/рамка"      андроид:layout\_marginEnd="16dp"      андроид:layout\_marginTop="76dp"      андроид:кликабельно="правда"      для Android:contentDescription="@NULL"в      андроид:срц="@холст/ic\_grade\_white"      приложение:fabSize="мини" />    </Уровень мельче> |

Если мы взглянем на это в окне предварительного просмотра мы видим, что это, конечно, делает то, что нужно, но жестко ДП ценности, которые на самом деле рассчитаны с высоты другой *вид* и Фабс внутренняя высота (которая сама может измениться, если мы изменим образ, или перетяжка) - это уже очень плохой код запах, и это привело к тому, что в макете, который будет не просто поддерживать:

[](https://i0.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/07/Hacky-FAB-cropped.png)

Так мы можем сделать это лучше в *ConstraintLayout*? Конечно, мы можем! Мы уже видели, что мы можем центр *вид* в родительском путем добавления ограничений относительно родителя, и что мы можем ограничить *вид* на другой *вид*. Но мы можем добавить несколько ограничений на единую опорную точку на другой *вид* и это позволяет нам сдерживать ФАБ до нижнего края панели:

Вещь, чтобы отметить здесь заключается в том, что у нас есть ограничения из верхней и нижней опорных точек ЛИВЕРПУЛЬСКОЙ которая атташе до нижней точки привязки панели, что приводит к ФАБ должно быть отцентрировано по вертикали на данный момент. Конечно, мы можем применить смещение, а также если мы хотим компенсировать это, но мы, собственно, хотим его центру, поэтому без предвзятости:

[](https://i2.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/07/ConstraintLayout-FAB-cropped.png)

Так что это уже намного чище, более ремонтопригодны решения, но он также решает еще один вопрос – мы можем установить панели высотой до wrap\_content динамического размера, чтобы высота его детей. В первом примере сокс это просто не сработает, и нам нужно переключиться на Java, чтобы создать пользовательский макет или, по крайней мере, позиция ФАБ программно после высоты панели были измерены. В *ConstraintLayout* решение, мы на самом деле узнать, что уже без необходимости менять!

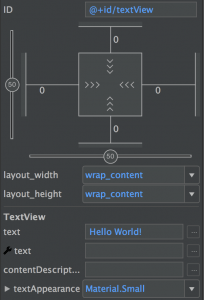
Надеюсь, пару простых примеров уже показать силу и гибкость, что *ConstraintLayout*предложения.

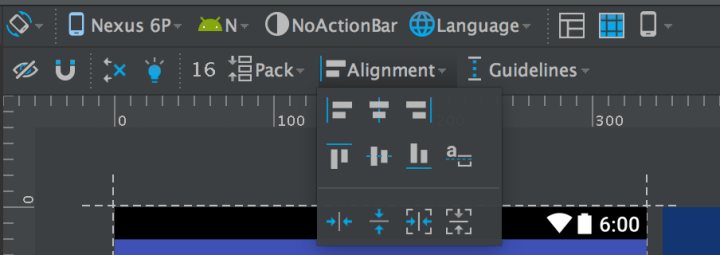
В последней статье в этой серии мы будем смотреть на пару другие функции визуального редактора, и рассуждать, где будующее *ConstraintLayout* может быть во главе.

Часть 7

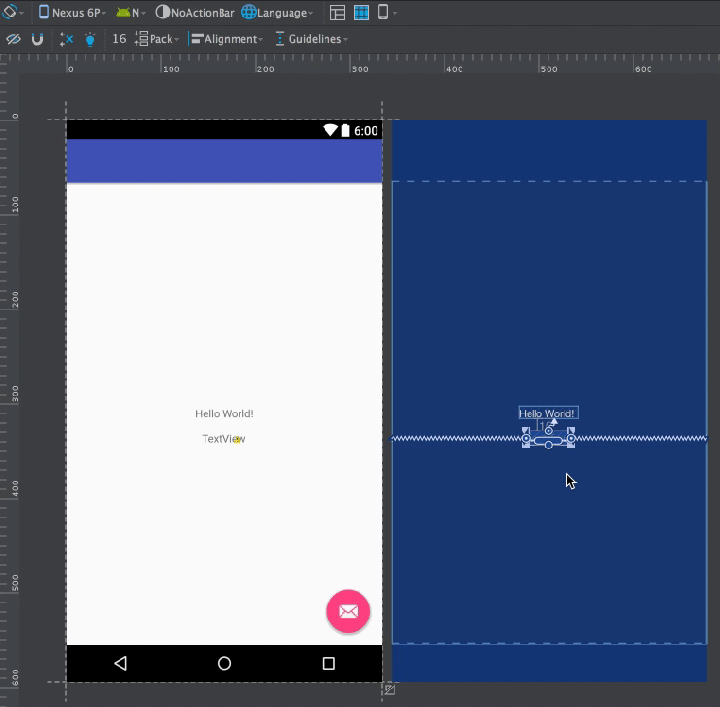
[15 июля 2016](https://blog.stylingandroid.com/constraintlayout-part-7/)[Марк Эллисон](https://blog.stylingandroid.com/author/admin/)[без комментариев](https://blog.stylingandroid.com/constraintlayout-part-7/#respond)

В Google ИО 2016 Google анонсировал новый Android-макета имени *ConstraintLayout*. Несмотря на первоначальный выпуск быть помечены как Альфа-релизе, это на самом деле очень стабильный (с некоторыми оговорками). В этой серии статей мы взглянем на этот новый макет-малыш на блоке и попытаться сделать некоторые выводы, как лучше использовать его.

[](https://blog.stylingandroid.com/constraintlayout-part-1/constraint-properties/)Пока в этой серии мы изучили многие аспекты *ContstraintLayout*себя, но есть пара особенностей визуального редактора, который также достойны исследования. Первый из них-это инструменты выравнивания, которые доступны из панели инструментов:

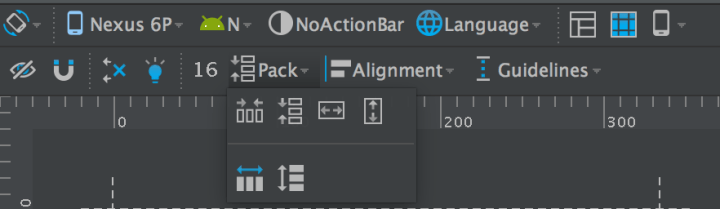
[](https://i1.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/07/alignment-tools.png)

В первой строке будет согласовывать все выбранные виджеты слева, справа или по центру друг от друга, и вторая строка будет делать эквивалент вертикально – с добавлением параметр, чтобы выровнять текстовые виджеты по базовой линии текста, а также. Третья линия-это все о центрирования элементов. Вторые два просто центр в макете родителей, но первые два являются немного более тонкими – они на самом деле центр в оставшееся пространство. В этом примере мы видим, что второй виджет изначально находится ниже первого, и когда мы выбираем вариант вертикальные центрирующие центры в оставшееся пространство из-за уже существующих ограничений на первых виджета:

[](https://i0.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/07/centre-vertical.gif)

Если мы остановиться и подумать о том, что все эти варианты на самом деле, они просто утилит, которые создают трудности для нас в определенной конфигурации, мы можем достичь всего этого сами, но это займет немного больше времени, чтобы создать все необходимые ограничения вручную. Если мы на самом деле смотрим на XML макеты (я не стану показывать их здесь, потому что там смотреть не на что) это просто установка ограничений.

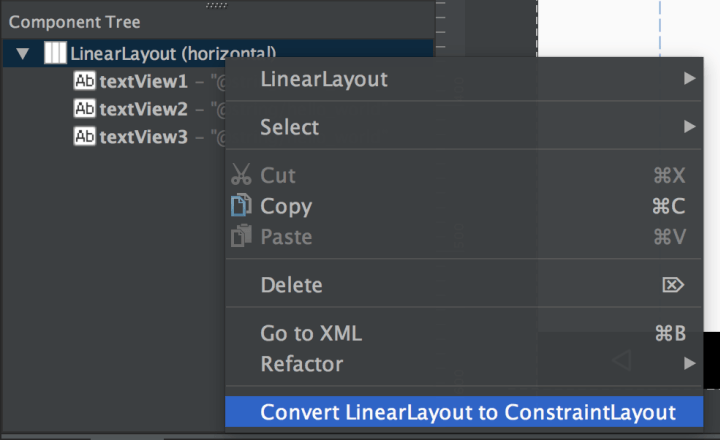
Другой инструмент-это варианты пакетов, которые, опять же, доступны из панели инструментов:

[](https://i2.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/07/pack-tools.png)

Те, кто знаком с графическими инструментами будем понимать здесь понятий. Первая пара пунктов, чтобы упаковать предметы в пределах доступного пространства по горизонтали или по вертикали – надо признать, что я не смог полностью понять такое поведение – это почти как работа. Вторая пара элементов будет расширяться выбранных элементов, чтобы заполнить родителя – это легче понять, но фактической реализации является жестко ДП значения вlayout\_[width|height] атрибуты в макете. Хотя это может показаться не проблемой во-первых, стоит учесть, что *ConstraintLayout* предназначен для работы с различными размерами и ориентациями экранов, так что хардкодить размеры просто идет против этого. Опять же, это чувствует, а как он по-прежнему работать в прогресс. Последняя пара предметов, распределять предметы по вертикали или по горизонтали – они работают немного лучше, потому что они на самом деле определить ограничения, а не использовать фиксированные значения ДП, поэтому единственный реальный пункт здесь что-то полезное в его нынешнем виде.

Дважды во время объяснения пакета инструменты я упомянул, что они чувствуют, что работа-в-прогресс, и я уверен, что они будут улучшаться с течением времени, но, сейчас, я бы не было соблазна использовать их для чего-либо другого, чем изучать, как они работают.

Окончательный инструмент, который стоит отметить-это инструмент для преобразования существующих макетов для ConstraintLayout. Это вызывается с панели "компоненты" дерево щелкнув правой кнопкой мыши на макет, который мы хотим преобразовать:

[](https://i2.wp.com/blog.stylingandroid.com/wp-content/uploads/2016/07/convert-tool.png)

В очередной раз это инструмент, который очень много работы в прогресс. Это действительно приятно видеть, что это идет но я должен сказать, что мой опыт общения с ней пока что не очень удачно – использование жестких ДП значения, и часто результат не особо близко к оригиналу макета. Однако, у меня есть большие надежды, что это позволит значительно улучшить в ближайшие релизы.

Это подводит нас к вопросу о миграции существующих макетов для *ConstraintLayout*. Если у вас есть работающие макеты, которые не страдают от проблем с производительностью, которые мы обсуждали ранее в этой серии, то есть мало получить путем преобразования в*ConstraintLayout*. Кроме того есть действительно случаи, когда простые макеты на самом деле будет более эффективным, используя другие форматы – например, если простой список элементов макета можно добиться, используя один уровень *макетом* , то это возможно будет сделать более эффективно, чем при использовании *ConstraintLayout*. Аналогично *заменить*только с одним из двух детей также оказывают гораздо более эффективно, чем *ConstraintLayout*. Главной мотивацией для переключения существующий макет для *ConstraintLayout* будет для целей уплощение иерархии и / или снижения стоимости проведения многократных измерений.

Так что мы можем ожидать в будущем? Очевидно, что мы увидим улучшения некоторых инструментов, которые мы рассмотрели в этой статье. Наиболее увлекательным из них является преобразование инструмент, потому что он фактически поддержал мой вывод двигатель, который пытается сопоставить существующий макет иерархии *ConstraintLayout*. В то время как это в настоящее время связана с преобразованием существующих макетов Android, имея такой двигатель, безусловно, обеспечит возможности для преобразования, а также другие форматы. Возможность импортировать, например, эскиз раскладки прямо в Android Studio-это интересная перспектива.

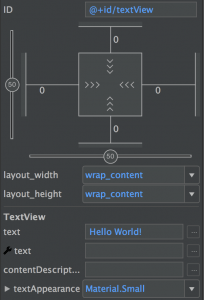
Одна наиважнейшая ощущение, что я играл с *ConstraintLayout* и, в частности, визуальный редактор в том, что многие из метафор используются, как выравнивание, упаковки и распределения товаров были взяты из средств проектирования и будет, по крайней мере, поможет преодолеть языковые различия между дизайнерами и разработчиками. Разработчики смогут говорить дизайнеры, используя терминологию, что они уже понимают, в результате их использования в визуальном редакторе. Возможно, мы сможем увидеть автономная версия визуальный редактор, который дизайнеры могут использовать и предоставить разработчикам дизайна, как макетов, работающих на Android, а не проекты, которые затем должны быть истолкованы и реализованы разработчиком.

В последней статье в этой серии мы будем долго смотреть на реальный вопрос: мы должны использовать *ConstraintLayout* в наши производственные проекты?

Часть 8

[22 Июля, 2016](https://blog.stylingandroid.com/constraintlayout-part-8/)[Марк Эллисон](https://blog.stylingandroid.com/author/admin/)[1 Комментарий](https://blog.stylingandroid.com/constraintlayout-part-8/#comments)

В Google ИО 2016 Google анонсировал новый Android-макета имени *ConstraintLayout*. Несмотря на первоначальный выпуск быть помечены как Альфа-релизе, это на самом деле очень стабильный (с некоторыми оговорками). В этой серии статей мы взглянем на этот новый макет-малыш на блоке и попытаться сделать некоторые выводы, как лучше использовать его.

[](https://blog.stylingandroid.com/constraintlayout-part-1/constraint-properties/)В ходе этой серии мы довольно долго смотреть на различные аспекты *ConstraintLayout* получить правильное ощущение того, как это работает, и действительно, зачем он был создан, в первую очередь. Однако существует один принципиальный вопрос, который мы должны рассмотреть: если мы на самом деле использовать его в “реальных” проектов? Короткий ответ “нет”, но твердость, что “нет” уменьшается с каждым выпуском.

Давайте сначала рассмотрим, почему я говорил, что мы не должны на самом деле использовать это в производстве проекты: Ромен парень неоднократно заявлял, что он не готов еще, и поэтому он все еще помечен как Альфа-релиз. *ConstraintLayout* себя довольно стабильным, но это визуальный редактор, который чуть менее стабильный, так почему бы не использовать *ConstraintLayout*как редактор не работает на устройствах пользователей? Есть несколько основных причин:

Во-Первых: Производительность. [Вольфрам Ритмейер написал отличный пост](http://www.grokkingandroid.com/thoughts-on-constraintlayout-and-design-editor/) , где он исследует многие аспекты *ConstraintLayout* в том числе производительности и обнаружили, что это не тариф слишком хорошо в сравнение с *макетом* и *уровень мельче*. Однако это было основано на Альфа-релиз 3 библиотеки и, на момент написания, сейчас у нас Альфа-4, который решает много проблем с производительностью. Я не видел ни в какое сравнение данных, чтобы увидеть, насколько хорошо Альфа 4 сравнивает с альфа-3 Производительность-мудрый, но Николай Roard (один из ConstraintLayout разработчиков) говорит, что он должен работать гораздо лучше.

Вторая причина, по которой *ConstraintLayout* все еще находится в Альфа заключается в том, что некоторые функции и API не завершен. Одним из примеров этого является abilty, чтобы использовать определения пропорции для просмотра с помощьюapp:layout\_constraintDimensionRatio атрибута. Причина, по которой я не осветили в этой серии, потому что это пока (на момент написания) поддерживается визуальный редактор (предложенная методика редактирования *ConstraintLayout*) . Хотя это будет чрезвычайно полезная функция ConstraintLayout (и я буду уверен, чтобы должным образом охватить его в свое время) тот факт, что он до сих пор не всплыли в визуальном редакторе было бы предположить, что либо функции или API еще не завершен.

Одной из главных причин не использовать *ConstraintLayout* был решен. До тех пор, пока Студия Android 2.2 Альфа 5 *ConstraintLayout* был в комплекте с Android Studio в себя, и это сделало его хитрым, чтобы использовать при запуске сборки на CI сервер. Те со своими Artifactoryбыл или Nexus случаев может вручную загрузить *ConstraintLayout* библиотеки и сделать их доступными для ИЦ этак. Тех, кто не потребуется вручную установить их на локальный репозиторий м2 на самом ИЦ. Все это было много усилий, особенно для Альфа-версий, для которых мы получаем регулярные обновления.

Хорошая новость заключается в том, что с Android Студия Альфа 5 года *ConstraintLayout* был перемещен в ее собственный m2repository – тот же механизм, с помощью которого поддержка библиотеки распространяются. Это делает вещи намного легче для пользователей ки, как и большинство стран СНГ имеют механизмы, чтобы сохранить поддержку библиотек до современных и так *ConstraintLayout* теперь может быть до-до-даты точно таким же образом.

Так вот где мы оставим *ConstraintLayout*. По крайней мере, сейчас. Уже существует ужасно много, как о *ConstraintLayout* и я действительно с нетерпением жду, чтобы быть в состоянии использовать его в производственных проектах. Команда, работающая над ее делают удивительную работу, и полное освобождение не может прийти достаточно скоро для меня. И, конечно, если есть какие-то интересные дополнения, они будут покрыты здесь!