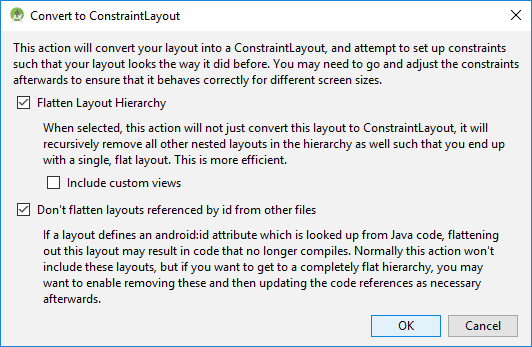
ConstraintLayout

Новый макет **ConstraintLayout** появился в Android Studio 2.2 и доступен для устройств с версии Android 2.3. Его можно найти в разделе **Layouts**. Гугл очень расхваливает данный макет и советует всем переходить на него и даже создал специальный конвертер для этой задачи. Если у вас имеется старый проект, то достаточно щёлкнуть правой кнопкой мыши на корневом элемента макета и выбрать пункт **Convert layout to ConstraintLayout**. В диалоговом окне ничего не трогаем.



В **build.gradle** модуля прописывается ссылка на библиотеку и проект начинает синхронизироваться. Сейчас уже активно развивается ветка 2.х.x, лучше сразу переходить на неё.

implementation 'com.android.support.constraint:constraint-layout:1.1.3'

// AndroidX

implementation 'androidx.constraintlayout:constraintlayout:1.1.3'

// ветка 2.х.x

implementation 'androidx.constraintlayout:constraintlayout:2.0.0-beta2'

В Android Studio 2.3 и выше в шаблонах по умолчанию теперь используется **ConstraintLayout**.

Если в режиме дизайна выбран данный компонент, то на панели инструментов вам доступны несколько кнопок.

* **View Options** с пунктами **Show Constraints** - выводит заданные ограничения в режимах предварительного просмотра и раскладки. В одних случаях этот режим просмотра полезен, в других нет. При большом количестве ограничений эта кнопка выводит слишком много информации, **Show Margins**, **Fade Unselected views**.
* **Turn On Autoconnect** - при включении режима Autoconnect ограничения будут автоматически настраиваться при перетаскивании представлений в область предварительного просмотра. Студия старается угадать, какие ограничения должен иметь компонент, и создавать их по мере необходимости
* **Default Margins** - стандартное значение для отступов. Можете устанавливать отдельно для каждого компонента. Выбрали компонент, установили значение, затем снова выбрали другой компонент и установили другое значение
* **Clear All Constraints** - удаляет все ограничения из макета
* **Infer Constraints** - автоматически создаёт ограничения. Срабатывает только при нажатии кнопки. Функциональность Autoconnect срабатывает каждый раз, когда в файл макета добавляется новый компонент
* **GuideLines** с двумя опциями: **Add Vertical GuideLine** и **Add Horizontal GuideLine**. Смотри ниже

Очень часто на форумах встречается вопрос, почему в режиме дизайна макет выглядит хорошо, а при запуске приложения все компоненты сбиваются в верхний левый угол. Для решения этой проблемы попробуйте нажать на кнопку **Infer Constraints**, которая создаст дополнительные ограничения.

**ConstraintLayout** является наследником **ViewGroup** и местами похож на **RelativeLayout**, но более продвинут. Код разметки в XML-представлении:

<?xml version="1.0" encoding="utf-8"?>

<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/activity\_main"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="ru.alexanderklimov.as22.MainActivity">

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginTop="16dp"

android:text="Hello World!"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintTop\_toTopOf="parent"

tools:layout\_constraintLeft\_creator="1"

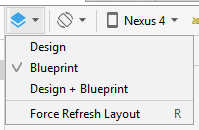
tools:layout\_constraintRight\_creator="1" />

</android.support.constraint.ConstraintLayout>

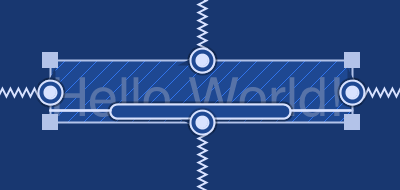
Управление компонентами внутри данного контейнера достаточно сильно отличается от старого взаимодействия. Придётся всем переучиваться.

При его использовании нет смысла использовать XML-представление, только в режиме **Design**, когда вы можете подвигать все компоненты в визуальном редакторе.

Рассмотрим отдельные элементы, которые используются для редактирования макета. Переключитесь в режим **Blueprint**, чтобы ничего нас не отвлекало от работы.

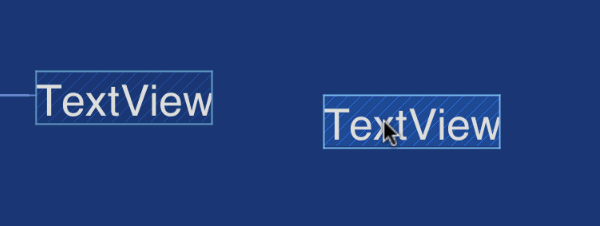


Так выглядит выбранный компонент.



Квадратные опорные точки в углах компонента позволяют изменять его размеры. Круглые опорные точки по краям позволяют управлять отступами от краёв экрана и других компонентов.

Продолговатый закругленный прямоугольник указывает на базовую линию текста. Пригодится при выравнивании по базовой линии другого компонента.



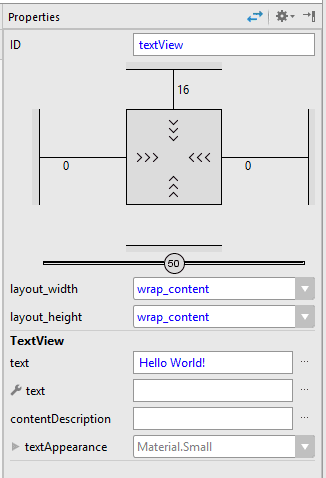
Также можно увидеть зигзагообразные направляющие-пружинки, которые нужны для особых случаев. Если провести мышкой над опорной точкой пружинки, она красиво активируется. Последующий щелчок удаляет пружинку. Если компонент удерживался двумя пружинками сверху и снизу, то удаление одной из них притянет компонент к верхней или нижней части экрана. Лучше самостоятельно проверить у себя, так как словами трудно описать.

Пружинки также можно легко восстановить, если щёлкнуть по круглой опорной точке и потянуть её к краю экрана.

Наверху на панели инструментов можно активировать автоматический режим размещения компонентов **Autoconnect** в виде магнита http://developer.alexanderklimov.ru/android/layout/constraintlayout10.png. В этом случае вы просто бросаете на экран нужный компонент, а далее студия сама пытается найти нужные параметры для него. Всё это происходит с красивой анимацией. Естественно, вы можете всё переиграть и поставить в нужную позицию.

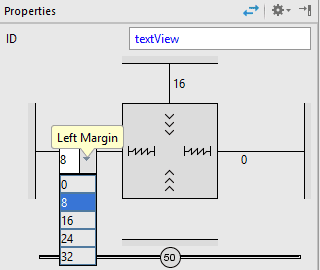
На той же панели есть инструмент **Infern Constraints** в виде пары звёздочек http://developer.alexanderklimov.ru/android/layout/constraintlayout11.png. Действует как **Autoconnect**, только работает не с одним компонентом, редактируемым в данный момент, а со всем макетом сразу, используя математические расчёты, чтобы определить, какие компоненты нужно привязать к другим, исходя из их местоположения на экране.

Теперь рассмотрим настройки в панели **Properties**.



Набор из трёх стрелок http://developer.alexanderklimov.ru/android/layout/constraintlayout7.png внутри квадрата означают атрибут **wrap\_content**. Если вы измените этот атрибут с помощью выпадающего списка или вручную напишите размер в **dp**, то увидите, что стрелки заменятся на прямые (фиксированный размер http://developer.alexanderklimov.ru/android/layout/constraintlayout8.png) или пружинку (**match\_constraints** http://developer.alexanderklimov.ru/android/layout/constraintlayout9.png, который является приблизительным аналогом атрибута **match\_parent** и имеет значение 0dp).

По бокам квадрата имеются числа. Если подвести к ним мышку, то появится выпадающий список с определёнными значениями: 0, 8, 16, 24, 32. Они отвечают за атрибут **margin** (отступы).

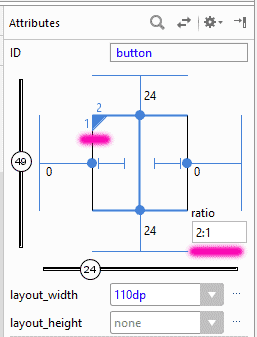


По бокам можно увидеть также ползунки с числами в кружочке. В примере виден только горизонтальный ползунок, при других настройках увидите и вертикальный. С их помощью можно центрировать компонент относительно экрана и других объектов.

Aspect ration (Соотношение сторон)

Если у компонента есть двусторонняя вертикальная привязка и значение высоты установлено в **match\_constraints (0dp)**, то можно настроить так, чтобы высота зависела от ширины. В углу появится треугольник, щёлкнув на котором можно затем установить желаемое соотношение. Потом можно изменять ширину, чтобы увидеть, как высота подстраивается под ширину.

Аналогично можно настроить зависимость ширины от высоты, предварительно сначала установив двустороннюю горизонтальную привязку.



GuideLine

На панели инструментов также имеется значок **GuideLines** с двумя опциями: **Add Vertical GuideLine** и **Add Horizontal GuideLine**. Если ими воспользоваться, то в XML-файле появятся такие строчки:

<android.support.constraint.Guideline

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/guideline"

android:orientation="vertical"

**app:layout\_constraintGuide\_begin="72dp"**

tools:layout\_editor\_absoluteY="0dp"

tools:layout\_editor\_absoluteX="72dp" />

По сути, это **View**, размер которого 0, что соответствует **View.GONE**. На этапе разработки мы видим только полоски, а во время работы приложения ничего не видим. Данные элементы помогают разместить компоненты аккуратно относительно линии.

Направляющие пригодятся, если одни и те же значения отступов повторяются для нескольких компонентов. Направляющие можно указывать в **dp** от края экрана или задать в процентах от ширины экрана. Чтобы переключаться между разными режимами, вы можете нажать на круглый значок **Guideline**.

Не всегда с помощью визуального редактора можно добиться нужного результата, тогда нужно переключиться в XML-режим. Один из таких случаев описан в статье [Square Island: Constraint Layout: Icon Label Text](http://blog.sqisland.com/2016/10/constraint-layout-icon-label-text.html).

Если есть желание работать через XML, то следует запомнить очень много атрибутов, например, для выравнивания относительно друг друга:

* app:layout\_constraintStart\_toStartOf="@id/view"
* app:layout\_constraintLeft\_toLeftOf="@id/view"
* app:layout\_constraintEnd\_toEndOf="@id/view"
* app:layout\_constraintRight\_toRightOf="@id/view"
* app:layout\_constraintTop\_toTopOf="@id/view"
* app:layout\_constraintBaseline\_toBaselineOf="@id/view"
* app:layout\_constraintBottom\_toBottomOf="@id/view"
* app:layout\_constraintStart\_toEndOf="@id/view"
* app:layout\_constraintLeft\_toRightOf="@id/view"
* app:layout\_constraintEnd\_toStartOf="@id/view"
* app:layout\_constraintRight\_toLeftOf="@id/view"
* app:layout\_constraintTop\_toBottomOf="@id/view"
* app:layout\_constraintBottom\_toTopOf="@id/view"

Атрибут **app:layout\_constraintHorizontal\_bias** используется float-значения от 0 до 1, для выравнивания по оси.

Атрибут **app:layout\_constraintDimensionRatio="4:3"** указывает, что нужно использовать данные пропорции по высоте и ширине для данного компонента. Также встречается модификация атрибута **app:layout\_constraintDimensionRatio="H, 1:1"**.

Chains - Скованные одной цепью

Несколько элементов можно сковать одной цепью. Допустим, у нас есть три кнопки. Выделяем их и через контекстное меню выбираем **Center Horizontally**. Снизу у выбранных компонентов появится символ цепи, а между ними будут нарисована связь в виде цепей. Если последовательно щёлкать по значку цепи, то увидите, как кнопки будут центрироваться с разными стилями:

* **spread** - Свободное пространство равномерно распределяется между выбранными компонентами и краями родителя (например, экрана)
* **spread\_inside** - Крайние компоненты прижимаются к границам родителя, свободное пространство равномерно распределяется только между остальными компонентами
* **packed** - Свободное пространство равномерно распределяется между крайними компонентами и границами родителя. Вы можете использовать **margin** для отступов

За цепи отвечают стили.

* **app:layout\_constraintHorizontal\_chainStyle="spread"**
* **app:layout\_constraintVertical\_chainStyle="spread"**
* **app:layout\_constraintHorizontal\_chainStyle="spread\_inside"**
* **app:layout\_constraintVertical\_chainStyle="spread\_inside"**
* **app:layout\_constraintHorizontal\_chainStyle="packed"**
* **app:layout\_constraintVertical\_chainStyle="packed"**

Также можно присвоить кнопкам вес, когда одна кнопка может быть больше остальных, это поведение знакомо по **LinearLayout**. Для этих целей используются атрибуты

* **layout\_constraintHorizontal\_weight**
* **layout\_constraintVertical\_weight**

Как и в **LinearLayout**, чтобы использовать вес, надо поставить размер компонента в **0dp**.

На рисунке этот вариант представлен в третьем примере.

Проценты

Можно указывать значения ширины и высоты в процентах через атрибуты **layout\_constraintWidth\_percent**, **layout\_constraintHeight\_percent**. Все View-компоненты поддерживают данные атрибуты. Они позволяют ограничить компонент процентным значением в рамках всего доступного пространства. Например, мы хотим видеть кнопку, которая будет занимать 70% в рамках свободного для неё места.

<Button

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

app:layout\_constraintWidth\_percent="0.7" />

Barriers

**Barriers** - это виртуальный **View**, который используется как шаблон. Он применяется для нескольких компонентов неизвестного размера – если один из них увеличивается, то барьер подстроит размер остальных под наибольшую высоту или ширину. Барьеры могут быть вертикальными и горизонтальными и создаваться сверху, снизу, слева или справа от нужных элементов. Другие элементы будут подстраиваться.

app:barrierDirection="right"

app:constraint\_referenced\_ids="button,text\_view"

Барьеры полезны, когда вы создаёте локализованные строки или отображаете контент, созданный пользователем, размер которого вы не можете предсказать.

Groups

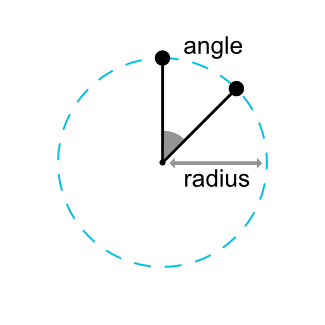
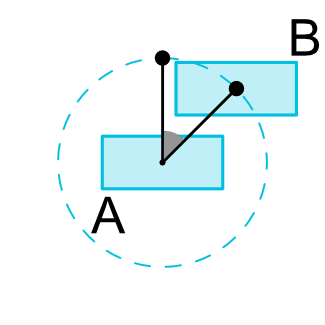
**Groups** - теперь можно логически группировать определённые виды. По сути, это некий контейнер, который содержит ссылки на ID компонентов, а не сами компоненты. При помощи группы вы можете установить видимость всех компонентов в контейнере. Это может пригодиться, когда сразу несколько элементов должны изменять свою видимость или другие свойства.

Анимация

Для анимации разметки **ConstraintLayout** используется **ConstraintSet**.

Circular

С помощью **Circular** мы можем настроить два компонента так, чтобы одно находилось на определённом расстоянии и под определённым углом от другого.

 <="" p="" style="box-sizing: border-box; vertical-align: middle; border-style: none;">

Потребуются атрибуты.

* **layout\_constraintCircle** - указываем идентификатор компонента, который будет центром окружности
* **layout\_constraintCircleRadius** - расстояние от центра окружности до компонента
* **layout\_constraintCircleAngle** - угол (в градусах, от 0 до 360)



При создании круговых зависимостей углы начинаются сверху и изменяются по часовой стрелке. Например, таким образом нужно описать среднюю кнопку в примере выше:

<android.support.design.widget.FloatingActionButton

android:id="@+id/middle\_expanded\_fab"

app:layout\_constraintCircle="@+id/fab"

app:layout\_constraintCircleRadius="50dp"

app:layout\_constraintCircleAngle="315" />

Такой способ пригодится для [анимации аналоговых часов](http://developer.alexanderklimov.ru/android/layout/constraintlayout_circular.php) или похожих примеров.

Placeholder

Элемент **Placeholder** позволяет использовать место на экране в качестве временного заполнителя. Используя анимацию, можно динамически перемещать компонент на место заполнителя.

Пример представлен в [отдельной статье](http://developer.alexanderklimov.ru/android/layout/constraintlayout_placeholder.php).

ConstraintLayoutStates (2.0.0)

В ветке 2.х появилась новая функциональность - переключение между макетами экрана. Это удобно, когда требуются небольшие изменения на одном экране. Вы должны создать несколько макет с *одинаковыми*компонентами, но при этом можете изменить их свойства (видимость, местоположение и т.д). В нужный момент программно переключаетесь на нужный макет.

Создадим три разных макета в папке **layout**.

res/layout/activity\_cl\_states\_start.xml

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/stateConstraintLayout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<ProgressBar

android:id="@+id/progressBarLoading"

style="?android:attr/progressBarStyle"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:visibility="invisible"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

<Button

android:id="@+id/button"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_marginBottom="8dp"

android:text="Start"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintHorizontal\_bias="0.0"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

res/layout/activity\_cl\_states\_loading.xml

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical">

<ProgressBar

android:id="@+id/progressBarLoading"

style="?android:attr/progressBarStyle"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

<Button

android:id="@+id/button"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_marginTop="8dp"

android:text="Start"

android:visibility="invisible"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintHorizontal\_bias="0.0"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toBottomOf="@+id/progressBarLoading" />

</androidx.constraintlayout.widget.ConstraintLayout>

res/layout/activity\_cl\_states\_end.xml

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:id="@+id/stateConstraintLayout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical">

<ProgressBar

android:id="@+id/progressBarLoading"

style="?android:attr/progressBarStyle"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:visibility="invisible"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

<Button

android:id="@+id/button"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:layout\_marginBottom="8dp"

android:text="Start"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintHorizontal\_bias="0.0"

app:layout\_constraintStart\_toStartOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

В нашем примере во всех макетах имеются **ProgressBar** и **Button** с разной видимостью и позицией.

Следующий шаг - создать в папке **res/xml** новый файл, описывающий три созданных макета.

res/xml/constraint\_layout\_states.xml

<?xml version="1.0" encoding="utf-8"?>

<ConstraintLayoutStates xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto">

<State

android:id="@+id/start"

app:constraints="@layout/activity\_cl\_states\_start"/>

<State

android:id="@+id/loading"

app:constraints="@layout/activity\_cl\_states\_loading"/>

<State

android:id="@+id/end"

app:constraints="@layout/activity\_cl\_states\_end"/>

</ConstraintLayoutStates>

Осталось написать программную часть. Сначала мы загружаем первый макет в стандартном методе **setContentView()**. Затем загружаем описания созданных нами макетов через **loadLayoutDescription()** объекта-контейнера нашего **ConstraintLayout**. Теперь мы можем переключаться между макетами через **constraintLayout.setState()**.

В примере при нажатии кнопки мы имитируем долгоиграющую задачу и выводим **ProgressBar**, когда задача будет завершена, то выводим третий макет, где кнопка находится уже в другом месте.

package ru.alexanderklimov.hellokot

import android.os.Bundle

import android.os.Handler

import androidx.appcompat.app.AppCompatActivity

import androidx.core.os.HandlerCompat.postDelayed

import kotlinx.android.synthetic.main.activity\_cl\_states\_end.\*

class MainActivity : AppCompatActivity() {

private val handler = Handler()

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_cl\_states\_start)

stateConstraintLayout.loadLayoutDescription(R.xml.constraint\_layout\_states)

var changed = false

button.setOnClickListener {

stateConstraintLayout.setState(R.id.loading, 0, 0)

postDelayed(handler, {

stateConstraintLayout.setState(if (changed) R.id.start else R.id.end,0, 0)

changed = !changed

}, null, 3000L)

}

}

}