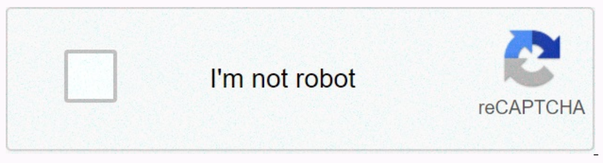

Continue

Get bitmap size android

Images are typically displayed using the built-in image view. This view takes care of loading and optimizing the image, freeing you to focus on app-specific details like layout and content. In this guide we will take a look at how to use an ImageView, how to manipulate bitmaps, learn about the different density folders and more. Using At the simplest level, an ImageView is simply a view that you embed in an XML layout that is used to display an image (or any drawable) on the screen. ImageView looks like this in res/layout/activity_main.xml: <ImageView android:id=@+id/image android:layout_width=wrap_content android:layout_height=wrap_content android:scaletype=center android:src=@drawable/my_image></ImageView> ImageView handles all loading and scaling the image for you. Note the scaleType attribute, which defines how the images are scaled to fit your layout. In the example using scaleType center, the image is displayed at the original resolution and centered in the view, no matter how much space the view uses. Sizing ImageView controls By default, the content of an ImageView control is of a certain size — usually the size of the image dimensions. They can also be bounded by their layout_width and layout_height attributes: <ImageView android:layout_width=50dp android:layout_height=50dp android:scaletype=fitXY...=></ImageView> ScaleType above is set to fitXY, which sets the height and up or down to fit the maximum specified dimensions. However, determining the width and height means that the proportions of the width and height of the original image, known as the height ratio, will be changed. We can take advantage of the AdjustViewBounds parameter to maintain this aspect ratio. However, we must either allow the height and/or width to be adjusted (i.e. by using maxWidth and using wrap_content for the dimension). Otherwise, the dimensions cannot be adjusted to meet the required aspect ratio. <ImageView android:layout_width=50dp android:layout_height=wrap_content android:scaletype=fitXY android:adjustviewbounds=true ...=></ImageView> By combining these properties together, we can control the rough size of the image and still adjust the image according to the correct aspect ratio. We can also size an ImageView when running within our Java source code by changing the width or height inside getLayoutParams () for the view: imageView.getLayoutParams ().height = 100; OR imageView.getLayoutParams().width = 100; imageView.getLayoutParams().height = 100 // OR imageView.getLayoutParams().width = 100 In some cases, the image is scaled to fit the width of the parent view and the height must be adjusted proportionally. We can achieve this using an extended ResizableImageView class as described in the post. Scale Types An ImageView can display an image based on scaleType provided. Above, we discussed the fitXY type with adjustViewBounds to match the aspect ratio of the one that can be pulled. Den Den is a list of all the most common types: Scale Type Description Center Displays the image centered in the view without scaling. centerCrop scales the image so that both the x and y dimensions are larger than or equal to the view while maintaining the image aspect ratio. centers the image in the view. centerInside Scales the image to fit into view while preserving the image format. If the image is already smaller than the view, it is the same as centered. fitCenter scales the image to fit into view while maintaining the image format. At least one axis corresponds exactly to the view, and the result is centered inside the view. fitStart Same as fitCenter, but aligned to the upper-left corner of the view. fitEnd Same as fitCenter, but aligned at the bottom right of the view. fitXY scales the x and y dimensions to exactly match the display size. does not retain the image format. array Scales the image using an included Matrix class. The matrix can be delivered using the setImageMatrix method. An Matrix class can be used to apply transformations, such as note: The FitXY scale type allows you to set the exact size of the image in your layout. However, be aware of potential distortions of the image due to scaling. If you create a picture display program, you are likely to use the Center or FitCenter scale types. For more information, see this ImageView ScaleType visual guide. Keep in mind that if you want to match the aspect ratio of the actual drawable, adjustViewBounds = true must be declared along with not defining an explicit width and/or height. Multis density support Since Android has so many different screen sizes, resolutions, and densities, there's a powerful system for choosing the correct image asset for the correct device. There are specific draw folders for each device density category, including: ldpi (low), mdpi (medium), hdpi (high), and xhdpi (extra high). Note that each app has folders for image draws such as drawable-mdpi, which is for medium dots per inch. To create alternate bitmap draws for different densities, follow the 3:4:6:8 scaling ratio between the four general densities. See chart below: Density DPI Sample Device Scale Pixels ldpi 120 Galaxy Y 0.75x 1dp = 0.75px mdpi 160 Galaxy Tab 1.0x 1dp = 1px hdpi 240 Galaxy S II 1.5 x 1dp = 11.5px xhdpi 320 Nexus 4 2.0x 1dp = 2px xxhdpi 480 Nexus 5 3.0x 1dp = 3px xxxhdpi 640 Nexus 6 4.0x 1dp = 4px That means, that if you generate a 100x100 for mdpi (1x baseline), then you must generate the same resource in 150x150 for hdpi (1.5x), 200x200 image for xhd 2.0x, 300 x 300 images for xxhdpi (3.0x) and 75x75 image for ldpi devices (0.75x). See these density guidelines for more information. Finally Android Resizer To resize images so check out the final Android Resizer by downloading and running this JAR. This handy utility allows us to choose a resources resources select an extra high density image and the tool will automatically generate the corresponding lower size images for us and place subfolders inside the generated res-drawable library within the actual res folder in your project as the example shows below in Project view (left) and standard Android view (right): Look to the screens support reference for a more detailed look at supporting a wide range of devices. See also the iconography guide for more information. Mipmaps and Drawables Starter with Android 4.3, there is now an option to use res/mipmap folder to store mipmap images. Mipmaps are most often used for application icons as the startup icon. To learn more about the benefits of mipmaps be sure to check the mipmapping for the drawables post. Mipmap image resources can then be accessed using @mipmap/ic_launcher instead of @drawable. Placement of icons in mipmap folders (instead of drawable) is considered a best practice because they can often be used in resolutions that are different from the current density of the device. For example, an xxxhdpi app icon might be a game icon. Review this post on preparing for nexus 6, which explains in more detail. Working with bitmaps We can change the bitmap displayed in an ImageView to a drawing resource with: ImageView image = (ImageView) findViewById(R.id.test_image); image.setImageResource(R.drawable.test2) val image = findViewById(R.id.test_image) as ImageView image.setImageResource(R.drawable.test2) or for arbitrary bitmap with: ImageView image = (ImageView) findViewById(R.id.test_image); Bitmap bMap = BitmapFactory.decodeFile(/sdcard/test2.png); image.setImageBitmap(bMap); val image = findViewById(R.id.test_image) as ImageView val bMap = BitmapFactory.decodeFile(/sdcard/test2.png) image.setImageBitmap(bMap) Scaling a bitmap If we need to resize a bitmap, we can call [createScaledBitmap]((android.graphics.Bitmap, int, int, boolean)] method of resizing a bitmap to our desired width and height: // Load a bitmap from the drawable folder Bitmap bMap = BitmapFactory.decodeResource(getResources(), R.drawable.my_image); Resize the bitmap to 150x100 (width x height) Bitmap bMapScaled = Bitmap.createScaledBitmap(bMap, 150, 100, true); Loads the size that has been split into an ImageView image = (ImageView) findViewById(R.id.test_image). image.setImageBitmap(bMapScaled); Load a bitmap from the small Bitmap bMap folder = BitmapFactory.decodeResource(getResources(), R.drawable.my_image); Resize the bitmap to 150x100 (width x height) Bitmap bMapScaled = Bitmap.createScaledBitmap(bMap, 150, 100, true); You often want to resize a bitmap, but maintain the aspect ratio by using a BitmapScaler utility class with code like this: The BitmapScaler public class { // Scale and maintain considering the desired width // // 100); public static bitmapscaleToFitWidth(Bitmap b, int width) { float factor = width / (float) b.getWidth(); return Bitmap.createScaledBitmap(b, width, (int) (b.getHeight() * factor), true); } // Scale and maintain aspect ratio given a desired height // Bitmapscaler.scaleToFitToHeight(bitmap, 100); public static bitmapscaleToFitHeight(Bitmap b, int height) { float factor = height / (float) b.getHeight(); return Bitmap.createScaledBitmap(b, (int) (b.getWidth() * factor), height, sand); } // ... object BitmapScaler { // Scale and maintain aspect ratio given the desired width // BitmapScaler.scaleToFitWidth(bitmap, 100); fun scaleToFitWidth(b: Bitmap, width: Int): Bitmap { val factor = width / b.width.toFloat() return Bitmap.createScaledBitmap(b, width, (b.height * factor).toInt(), true) } // Scale and maintenance height format given a desired height // BitmapScaler.scaleToHeight(bitmap, 100); fun scaleToFitHeight(b: Bitmap, height: Int): Bitmap { val factor = height / b.height.toFloat() return Bitmap.createScaledBitmap(b, (b.width * factor).toInt(), height, true) } // ... In other cases, you may want to determine the height or width of the device to resize the image accordingly. Copy this DeviceDimensionsHelper.java Tool Class for DeviceDimensionsHelper.java in your project, and use any location you have context to determine the screen dimensions: // Get the height or width of the screen at runtime int screenWidth = DeviceDimensionsHelper.getDisplayWidth(this); Resize a bitmap system that maintains aspect ratio based on the BitmapScaler.scaleToFitWidth(bitmap, screenWidth); Get the height or width of the screen when running int screenWidth = DeviceDimensionsHelper.getDisplayWidth(this); Resize a bitmap system that maintains aspect ratio based on the BitmapScaler.scaleToFitWidth(bitmap, screenWidth); See this source for more information about how to scale a bitmap based instead on the relative unit width and height. Note: If you make any type of scaling of images, loss of EXIF metadata occurs, which includes information such as camera, rotation, date/time of the photo taken. There are solutions to transferring this data after the image has been copied, but there are current limitations. If you need this information or want to upload it to a website, send the original file and not the disguising version. Viewing SVG Images Android now has vector drawables support which allows SVG files to be imported to a specific format. SVG files can be automatically converted using Android Studio by going to File -> New -> Vector Asset. Be sure to click Local File (SVG, PSD) to import the file. References References