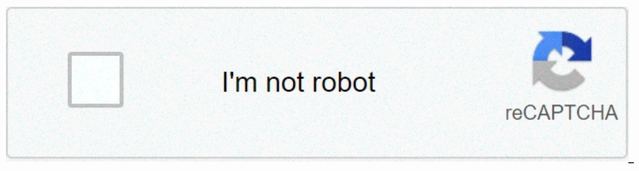

Continue

Android. graphics. bitmap to byte array c

I have an android.graphics.bitmap and an android.net.Uri both of these I can use anyway I want. But I don't know how to take bitmaps and convert it to bye[] I've tried to use a parcel, but I can't initial it, and when I use it in the writetoparcel method for both bitmap and uri, it throws errors. I tried bitmaps the ToArray method and that did nothing but create an empty array. I also tried to use the compression method, but I can't initialize a stream. The text editor throws an error about creating a new stream inside an abstract class. Is there some reference that I missed that allows me to do so. class Bitmap : Parcelable kotlin. Any ↳.graphics.Bitmap Nested Class CompressFormat specifies known bitmap formats can be compressed into the possible Bitmap configuration. Constants static Int DENSITY_NONE Indicates that the bitmap was created for an unknown pixel density. Inherited constants from the Parcelable Int CONTENTS_FILE_DESCRIPTOR Descriptor bit class used with describContents(): indicate that the flat representation of the closed object contains a file descriptor. Int PARCELABLE_WRITE_RETURN_VALUE Flag for use with writeToParcel: the object being written is a return value, that is the result of a function such as Parcelable someFunction(), void someFunction(out Parcelable), or void someFunction(inout Parcelable). Some implementations may want to free up resources at this point. General Boolean compression methods(Format: Bitmap.CompressFormat!, Quality: Int, Stream: OutputStream!) write a compressed version of bitmap to the specified output stream. Bitmap! copy(config: Bitmap.Config!, isMutable: Boolean) Tries to make a new bitmap based on the dimensions of this bitmap, setting the new bitmap's config to the one specified, and then copying this bitmap's pixels into the new bitmap. CopyPixelsFromBuffer unit (src: buffer!) copy pixels from the buffer, which starts from the current position and rewrites bitmap pixels. CopyPixelsToBuffer(dst: Buffer!) unit copies bitmap pixels into the specified buffer (assigned by the caller). Static Bitmap! CreateBitmap (src: Bitmap) returns a bitmap from the bitmap source. Static Bitmap! createBitmap(source: Bitmap, x: Int, y: Int, width: Int, height: Int) returns a bitmap from the specified subset of the source bitmap. Static Bitmap! createBitmap(source: Bitmap, x: Int, y: Int, width: Int, height: Int, m: Matrix?, filter: Boolean) Returns a bitmap from subset of the source bitmap, transformed by the optional matrix. Static Bitmap! createBitmap(width: Int, height: Int, config: Bitmap.Config) Returns a mutable bitmap with the specified width and height. Static Bitmap! createBitmap(display: DisplayMetrics?, width: Int, height: Int, config: Bitmap.Config) returns a mutable bitmap with the specified width and height. Static Bitmap! createBitmap(width: Int, height: Int, config: Bitmap.Config, hasAlpha: Boolean) Returns a mutable bitmap with the specified width ارتفاع. استاتیک Bitmap! createBitmap(width: Int, height: Int, config: Bitmap.Config, hasAlpha: Boolean, colorSpace: ColorSpace) Returns a mutable bitmap with the specified width and height. استاتیک Bitmap! createBitmap(display: DisplayMetrics?, width: Int, height: Int, config: Bitmap.Config, hasAlpha: Boolean) Returns a mutable bitmap with the specified width and height. استاتیک Bitmap! createBitmap(display: DisplayMetrics?, width: Int, height: Int, config: Bitmap.Config, hasAlpha: Boolean, colorSpace: ColorSpace) returns a mutable bitmap with the specified width and height. استاتیک Bitmap! createBitmap(colors: IntArray, offset: Int, stride: Int, width: Int, height: Int, config: Bitmap.Config) Returns a immutable bitmap with the specified width and height, with each pixel value set to the corresponding value in the colors array. استاتیک Bitmap! createBitmap(display: DisplayMetrics, colors: IntArray, offset: Int, stride: Int, width: Int, height: Int, config: Bitmap.Config) returns a immutable bitmap with the specified width and height, with each pixel value set to the corresponding value in the colors array. استاتیک Bitmap! createBitmap(رنگ: IntArray, عرض: Int, ارتفاع: Int, پیکربندی: Bitmap.Config!) استاتیک یک بیت مپ تغییر ناپذیر با عرض و ارتفاع مشخص شده، با هر مجموعه مقدار پیکسل به مقدار مربوطه در آرایه رنگ ها برمی گردد. Bitmap! createBitmap(display: DisplayMetrics?, colors: IntArray, width: Int, height: Int, config: Bitmap.Config) Returns a immutable bitmap with the specified width and height, with each pixel value set to the corresponding value in the colors array. bitmap ایجادBitmap(ثابت منبع: تصویر) ایجاد یک Bitmap از منبع تصویر داده شده از دستورات نقاشی ثبت شده است. static Bitmap createBitmap(source: Picture, width: Int, height: Int, config: Bitmap.Config) Creates a Bitmap from the given Picture source of recorded drawing commands. استاتیک Bitmap! createScaledBitmap(src: Bitmap, dstWidth: Int, dstHeight: Int, filter: Boolean) Creates a new bitmap, scaled from an existing bitmap, when possible. Int describContents() No special parcel contents. Unit eraseColor(c: Int) پیکسل های بیت مپ را با رنگ مشخص شده پر می کند. پاک کردن رنگ واحد(رنگ: بلند) پیکسل بیت مپ را با ColorLong مشخص شده پر می کند. Bitmap! extractAlpha() بازگشت بیت مپ جدید است که گرفتن مقادیر آلفا از اصلی. Bitmap! extractAlpha(رنگ: رنگ!, offsetXY: IntArray!) بیت مپ جدیدی را برمیگرداند که مقادیر آلفای اصلی را ضبط می کند. Int getAllocationByteCount() اندازه حافظه اختصاص داده شده مورد استفاده برای ذخیره پیکسل های این بیت مپ را برمیگرداند. Int getByteCount() رنگ. حداقل تعداد بایت هایی را برمیگرداند که می توانند برای ذخیره پیکسل های این بیت مپ مورد استفاده قرار گیرند. getColor(x: Int, y: Int) رنگ را در محل مشخص شده برمیگرداند. فضای رنگی؟ getColorSpace() فضای رنگی مرتبط با این بیت مپ را برمیگرداند. Bitmap.Config! getConfig() اگر پیکربندی داخلی bitmap در یکی از فرمت های عمومی است، بازگشت که پیکربندی، در غیر این صورت بازگشت null. هوشمند Returns the density of this bitmap. Int getGenerationId() ID returns the generation of this bitmap. Int getHeight() Returns the bitmap's height ByteArray! getNinePatchChunk() returns an optional array of private data, used by the UI system for some bitmaps. Int getPixel(x: Int, y: Int) returns the color at the specified location. Unit getPixels(pixels: IntArray!, offset: Int, stride: Int, x: Int, y: Int, width: Int, height: Int) Returns in pixels[] a copy of the data in the bitmap. Int getRowBytes() returns the number of bytes between rows in bitmap pixels. Int getScaledHeight (Canvas: Boom!) is given comfort to call getScaledHeight (int) with canvas target density. Int getScaledHeight (metrics: DisplayMetrics!) is given comfort to call getScaledHeight (int) with target density from DisplayMetrics. Int getScaledHeight(targetDensity: Int) Convenience method that returns the height of this bitmap divided by the density scale factor. Int getScaledWidth (Boom: Boom!) is the convenience of calling getScaledWidth (int) with the target density of the canvas. Int getScaledWidth (metrics: DisplayMetrics!) is easy to call getScaledWidth(int) with target density from DisplayMetrics. Int getScaledWidth(targetDensity: Int) Convenience method that returns the width of this bitmap divided by the density scale factor. Int getWidth() returns the bitmap's width Boolean hasAlpha() Returns true if the bitmap's config supports per-pixel alpha, and if the pixels may contain non-opaque alpha values. Boolean hasMipMap() indicates whether the rendering responsible for drawing this bitmap should be an attempt to use mipmaps when this bitmap scales down. Boolean isMutable() Returns true if the bitmap is marked as mutable (i.e. can be drawn into) Boolean isPremultiplied() Indicates whether pixels stored in this bitmaps are stored pre-multiplied. Boolean isRecycled() Returns true if this bitmap has been recycled. Unit prepareToDraw() Builds caches associated with the bitmap that are used for drawing it. The Reconfiguration Unit (Width: Int, Height: Int, Configuration: Bitmap.Config!) modifies bitmap to have a certain width, height and configuration, without affecting the allocation of the field that supports bitmap. Recycle unit() free native object associated with this bitmap, and turn on reference to pixel data. Boolean sameAs(other: Bitmap!) according to another bitmap, return just if it has the same dimensions, configuration, and pixel data as this bitmap. The setColorSpace unit (colorSpace: ColorSpace) corrects the bitmap to have the specified ColorSpace, without affecting the allocation of the field that supports bitmap. SetConfig unit (configuration: Bitmap.Config!) is an easy way to contact reconfiguration (int,int,android.graphics.Bitmap.Config) with current height and width. The setDensity unit (density: Int) determines the density of this bitmap. Unit setHasAlpha(hasAlpha: Boolean) Tell the bitmap if all of the pixels are known to be opaque (false) or Some pixels may contain non-opaque (true) alpha values. The setHasMipMap (hasMipMap: Boolean) unit set a hint for the rendering responsible for drawing this bitmap indicates that it should attempt to use mipmaps when this bitmap scale is pulled down. SetHeight unit (Height: Int) is an easy way to contact reconfiguration (int,int,android.graphics.Bitmap.Config) with current width and configuration. Unit setPixel(x: Int, y: Int, color: Int) Write the specified Color into the bitmap (assuming it is mutable) at the x,y coordinate. Unit setPixels (Pixel: IntArray!, Offset: Int, Step: Int, x: Int, y: Int, width: Int, height: Int: Int) Replace pixels in the bitmap with the colors in the array. The setPremultiplied unit (premultiplied: Boolean) sets whether Bitmap should treat its data as pre-multiplied. Unit setWidth(width: Int) Convenience method for calling reconfigure(int,int,android.graphics.Bitmap.Config) with the current height and config. Static Bitmap? WrapHardwareBuffer (hardwareBuffer: HardwareBuffer, colorSpace: ColorSpace?) create a hardware bitmap with the support of a HardwareBuffer. Unit writeToParcel(p: Parcel!, flags: Int) Write the bitmap and its pixels to the parcel. Properties static Parcelable.Creator<Bitmap!> fun compress(format: Bitmap.CompressFormat!, quality: Int, stream: OutputStream!): Boolean Write a compressed version of the bitmap to the specified outputstream. If this return is correct, bitmap can be restored by passing a corresponding input stream to BitmapFactory.decodeStream(). Note: All formats support all bitmap configurations directly, so it is possible that bitmap returning from BitmapFactory can vary in bitdepth, and/or may have been lost per alpha pixel (as . JPEG only supports opaque pixels). This method may take a few seconds to complete, so it should only be called from a worker yarn. Parameters Bitmap.CompressFormat Format!: Int Compressed Image Quality Format: Refers to Compressor, 0-100. The value is interpreted differently depending on CompressFormat. stream OutputStream!: The outputstream to write the compressed data. Boolean's return is marked just if successfully compressed into the stream. Fun Copy (Configuration : Bitmap.Config!, isMutable: Boolean): Bitmap! It tries to set a new bitmap based on the dimensions of this bitmap, configure the new bitmap to the specified one, and then copy the pixels of this bitmap to the new bitmap. If the conversion is not supported, or the allotment fails, then this NULL will return. The returned bitmap has the same density and color space as the original, except in the following cases. When copying to Config#ALPHA_8, the color space is released. When copying or from Config#RGBA_F16, EXTENDED or non-EXTENDED may be set up appropriately. Parameters Configuration Bitmap.Config!: The configuration for the resulting bitmap isMutable Boolean: It's true if the resulting bitmap should be mutated (as one of its pixels can be modified) Bitmap return! New Bitmap, </Bitmap!>null if the copy cannot be made. Exceptions java.lang.IllegalArgumentException if config is Config#HARDWARE and isMutable is true fun copyPixelsFromBuffer(src: Buffer!): Unit Copy the pixels from the buffer, beginning at the current position, overwriting the bitmap's pixels. The data in the buffer does not change at all (unlike setPixels() which is converted from 32bit without prediction to any native bitmap format. Exceptions java.lang.IllegalStateException if the bitmap's config is Config#HARDWARE fun copyPixelsToBuffer(dst: Buffer!): Unit Copy the bitmap's pixels into the specified buffer (allocated by the caller). An exception is thrown if the buffer is not large enough to hold all pixels (taking into light of the number of bytes per pixel) or if the buffer subclass is not one of the types of support (ByteBuffer, ShortBuffer, IntBuffer). Bitmap content is copied to the as-is buffer. This means that if this bitmap saves your pixels in pre-multiplication (see isPremultiplied() values in the buffer will also be multiplied in the pre-multiplication. Pixels remain in the bitmap color space. The new bitmap may be the source object, or it may be a copy made with the same density and color space as the original bitmap. : This value cannot be empty. Static Fun CreateBitmap (Source: Bitmap, x: Int, y: Int, Width: Int, Height: Int): Bitmap! Returns a bitmap from the specified bitmap subset of the source. The new bitmap may be the same as the source object, or a copy may have been made. With the same density and color space, the original bitmap is initial. Parameters Bitmap Source: Our bitmap below setting this value cannot be empty. x Int: First x pixel coordinates in source y Int: y pixel coordinates first in int source width: number of pixels per row Height Int: Bitmap return row number! A copy of the source bitmap subset or the source bitmap itself. Exceptions java.lang.IllegalArgumentException if the x, y, width, height values are outside of the dimensions of the source bitmap, or width is <= 0, or height is <= 0 static fun createBitmap(source: Bitmap, x: Int, y: Int, width: Int, height: Int, m: Matrix?, filter: Boolean): Bitmap! Returns a bitmap from the source bitmap subset, which is converted by optional The new bitmap may be the same as the source object, or a copy may have been made. With the same density and color space, the original bitmap is initial. If the bitmap is an unchangeable source and the requested subset is the source bitmap itself, then the source bitmap will be returned and no new bitmaps will be created. The recursible bitmap will always be modifiable except in the following scenarios: (1) When the source bitmap is returned and the bitmap is an immutable source(2) the source bitmap is a hardware bitmap. That getConfig() is equivalent to Config#HARDWARE Parameters Source Bitmap: The bitmap we set below cannot be depleted. x Int: First x pixel coordinates in source y Int: y pixel coordinates first in int source width: number of pixels per row Int height: number of rows m matrix?: Optional matrix to apply to pixels This value may be null. Boolean Filter: It is true if the source should be filtered. Only applies if the matrix contains more than just translation. Bring back The Bethamp! A bitmap that represents the specified subset of exceptions java.lang.IllegalArgumentException if x, y, width, height values are outside the dimensions of the source bitmap, or the width <= is 0, or the height <= 0, or if the source bitmap is already recycled static fun createBitmapmap (width: Int, height: Int, configuration: Bitmap Returns a bit of mutantmap with the specified width and height. Its initial density is as high as any getDensity. The newly created Bitmap in ColorSpace.Named#SRGB is the color space. Parameters Width Int: Width of Bitmap Height Int: Bitmap Height Configuration Bitmap.Config: Bitmap Configuration to Create. This value cannot be depleted. Exceptions java.lang.IllegalArgumentException if the width or height are <= 0, or if Config is Config.HARDWARE, because hardware bitmaps are always immutable static fun createBitmap(display: DisplayMetrics?, width: Int, height: Int, config: Bitmap.Config): Bitmap! Returns a bit of mutantmap with the specified width and height. Its initial density is determined from the given DisplayMetrics. The newly created Bitmap in ColorSpace.Named#SRGB is the color space. Parameters displayMetrics?: Display metrics will be drawn to display this bitmap on it. This value may be empty. Width int: Bitmap width Int height: bitmap height Configuration Bitmap.Config: Configure bitmap to create. This value cannot be depleted. Exceptions java.lang.IllegalArgumentException if the width or height are <= 0, or if Config is Config.HARDWARE, because hardware bitmaps are always immutable static fun createBitmap(width: Int, height: Int, config: Bitmap.Config, hasAlpha: Boolean): Bitmap! Returns a bit of mutantmap with the specified width and height. Its initial density is as high as any getDensity. The newly created Bitmap in ColorSpace.Named#SRGB is the color space. Parameters Width Int: Width of Bitmap Int Height: Height Configure Bitmap.Config: Configure bitmap to create. This value cannot be depleted. hasAlpha Boolean: If the bitmap is ARGB_8888 or RGBA_16F this flag can be used to mark the bitmap as opaque. Doing so instead of being transparent, it will erase bitmaps in black. Exceptions java.lang.IllegalArgumentException if the width or height are <= 0, or if Config is Config.HARDWARE, because hardware bitmaps are always immutable static fun createBitmap(width: Int, height: Int, config: Bitmap.Config, hasAlpha: Boolean, colorSpace: ColorSpace): Bitmap! Returns a bit of mutantmap with the specified width and height. Its initial density is as high as any getDensity. Parameters Width Int: Width of Bitmap Height Int: Bitmap Height Configuration Bitmap.Config: Bitmap Configuration to Create. This value cannot be depleted. hasAlpha Boolean: If the bitmap is ARGB_8888 or RGBA_16F this flag can be used to mark the bitmap as opaque. Doing so instead of being transparent, it will erase bitmaps in black. ColorSpace ColorSpace: Colorspace of bitmap. If the configuration is configured #RGBA_F16 and ColorSpace.Named#SRGB or ColorSpace.Named#LINEAR_SRGB is provided, then the corresponding extended range type is assumed. This value cannot be depleted. Exceptions java.lang.IllegalArgumentException if the width or height are <= 0, if Config is Config.HARDWARE (because hardware bitmaps are always immutable), if the specified color space is not ColorSpace.Model#RGB, If the color space transfer function is not specified ColorSpace.Rgb.TransferParameters, or if the color space is empty static fun createBitmap (Screen: DisplayMetrics?, Width: Int, Height: Int, Configuration: Bitmap.Config, hasAlpha: Boolean): Bitmap! Returns a bit of mutantmap with the specified width and height. Its initial density is determined from the given DisplayMetrics. The newly created Bitmap in ColorSpace.Named#SRGB is the color space. Parameters

displayMetrics?: Display metrics will be drawn to display this bitmap on it. This value may be empty. Width int: Bitmap width Int height: bitmap height Configuration Bitmap.Config: Configure bitmap to create. This value cannot be depleted. hasAlpha Boolean: If the bitmap is ARGB_8888 or RGBA_16F this flag can be used to mark the bitmap as opaque. Doing so instead of being transparent, it will erase bitmaps in black. Exceptions java.lang.IllegalArgumentException if the width or height are <= 0, or if Config is Config.HARDWARE, because hardware bitmaps are always immutable static fun createBitmap(display: DisplayMetrics?, width: Int, height: Int, config: Bitmap.Config, hasAlpha: Boolean, colorSpace: ColorSpace): Bitmap! Returns a bit of mutantmap with the specified width and height. Its initial density is determined from the given DisplayMetrics. The newly created Bitmap in ColorSpace.Named#SRGB is the color space. Parameters displayMetrics?: Display metrics will be drawn to display this bitmap on it. This value may be empty. Width int: Width of int height bitmap: Bitmap height configuration Bitmap.Config: bitmap configuration to create. This value cannot be depleted. hasAlpha Boolean: If the bitmap is ARGB_8888 or RGBA_16F this flag can be used to mark the bitmap as opaque. Doing so instead of being transparent, it will erase bitmaps in black. ColorSpace ColorSpace: Colorspace of bitmap. If the configuration is configured #RGBA_F16 and ColorSpace.Named#SRGB or ColorSpace.Named#LINEAR_SRGB is provided, then the corresponding extended range type is assumed. This value cannot be depleted. Exceptions java.lang.IllegalArgumentException if the width or height are <= 0, if Config is Config.HARDWARE (because hardware bitmaps are always immutable), if the specified color space is not ColorSpace.Model#RGB, If the color space transfer function is not specified ColorSpace.Rgb.TransferParameters, or if the color space is empty static fun createBitmap (Color: IntArray, Offset: Int, Step: Int, Width: Int, Height: Int, Configuration: Bitmap.Config): Bitmap! An immutable bitmap with the specified width and height, with each set of pixel values returns to the corresponding value in the color array. Its initial density is as high as any getDensity. The newly created Bitmap in ColorSpace.Named#SRGB is the color space. IntArray Color Parameters: An array of sRGB colors used for pixel initials. This value cannot be depleted. Offset Int: Count values to skip before the first color in an array of colors. Step Int: Count colors in the array between rows (must >= width or <= -width). Width int: Bitmap width Int height: bitmap height Configuration Bitmap.Config: Configure bitmap to create. If the configuration does not support any alpha pixel (such as RGB_565), then alpha bytes are ignored in colors (presumed to be FF) this value cannot be depleted. Exceptions java.lang.IllegalArgumentException if the width or height are <= 0, or if the color array's length is less than the number of pixels. Static Fun CreateBitmap (Screen: DisplayMetrics, Color: IntArray, Offset: Int, Step: Int, Width: Int, Height: Int, Configuration: Bitmap.Config): Bitmap! An immutable bitmap with the specified width and height, with each set of pixel values returns to the corresponding value in the color array. Its initial density is determined from the given DisplayMetrics. The newly created Bitmap in ColorSpace.Named#SRGB is the color space. Parameters displayMetrics: Display metrics will be drawn to display this bitmap on it. This value cannot be depleted. IntArray Color: An array of sRGB colors used for primary pixels. This value cannot be depleted. Offset Int: Count values to skip before the first color in an array of colors. Step Int: Count colors in the array between rows (must >= width or <= -width). Width int: Bitmap width Int height: bitmap height Configuration Bitmap.Config: Configure bitmap to create. If the configuration does not support any alpha pixels (such as RGB_565) then alpha bytes are ignored in colors [] (assumed be FF) This value cannot be depleted. Exceptions java.lang.IllegalArgumentException if the width or height are <= 0, or if the color array's length is less than the number of pixels. Static Fun CreateBitmap (Color: IntArray, Width: Int, Height: Int, Configuration: Bitmap.Config!): Bitmap! An immutable bitmap with the specified width and height, with each set of pixel values returns to the corresponding value in the color array. Its initial density is as high as any getDensity. The newly created Bitmap in ColorSpace.Named#SRGB is the color space. IntArray Color Parameters: An array of sRGB colors used for pixel initials. This array must be at least as wide as * height. This value cannot be depleted. Int Width: Bitmap Width Int Height: Bitmap Height Configuration Bitmap.Config!: Configure bitmap to create. If the configuration does not support any alpha pixel (such as RGB_565), then alpha bytes will be ignored in colors (presumed to be FF) java.lang.IllegalArgumentException exceptions if the width or height <= 0, or if the length of the color array is less than the number of pixels. Static Fun CreateBitmap (Screen: DisplayMetrics?, Color: IntArray, Width: Int, Height: Int, Configuration: Bitmap.Config): Bitmap! An immutable bitmap with the specified width and height, with each set of pixel values returns to the corresponding value in the color array. Its initial density is determined from the given DisplayMetrics. The newly created Bitmap in ColorSpace.Named#SRGB is the color space. Parameters displayMetrics?: Display metrics will be drawn to display this bitmap on it. This value may be empty. IntArray Color: An array of sRGB colors used for primary pixels. This array must be at least as wide as * height. This value cannot be depleted. Width int: Bitmap width Int height: bitmap height Configuration Bitmap.Config: Configure bitmap to create. If the configuration does not support any alpha pixel (such as RGB_565), then alpha bytes are ignored in colors (presumed to be FF) this value cannot be depleted. Exceptions java.lang.IllegalArgumentException if the width or height are <= 0, or if the color array's length is less than the number of pixels. Static fun createBitmap (Source: Picture): Bitmap creates Bitmap from the source of the given image of the recorded drawing commands. CreateBitmap (android.graphics.Picture, int,int,android.graphics.Bitmap.Config) with the same width and height as the width and height of the picture and config.HARDWARE configuration. Parameters Image Source: The recorded image of the drawing commands that will be returned to Bitmap will be drawn. This value cannot be depleted. Return bitmap bits an immutable map by configuring the hardware that created its contents from the drawing commands recorded in the image source. This value cannot be depleted. Static Fun CreateBitmap (Source: Image, Width: Int, Height: Int, Configuration: Bitmap.Config): Bitmap Creates a Bitmap from the given image source of the recorded drawing Bitmap will be unchangeable with the width and height given. If the width and height are the same as the width and height of the picture, the image will be scaled to fit the width and height given. Parameters Image Source: The recorded image of the drawing commands that will be returned to Bitmap will be drawn. This value cannot be depleted. Width int: The width of the bitmap to create. The width of the scale image will match up if necessary. Height int: Bitmap height to create. The height of the scale image will match up if necessary. config Bitmap.Config: The Config of the created bitmap. This value cannot be depleted. The return of the immutable Bitmap bitmap with the configuration specified by the configuration parameter of this value cannot be depleted. Static Fun CreateScaledBitmap (src: Bitmap, dstWidth: Int, dstHeight: Int, Filter: Boolean): Bitmap! Creates a new bitmap that is scaled from an existing bitmap when possible. If the specified width and height are the same as the current width and height of the source bitmap, the source bitmap will be returned and a new bitmap will not be created. Parameters src Bitmap: Bitmap Source. This value cannot be depleted. dstWidth Int: The desired width of the new bitmap. dstHeight Int: the new bitmap's desired height. Boolean Filter: Whether or not a two-line filter should be used when scaling bitmap. If this is true then the two-line filter will be used when scaling is better at a worse performance cost. If this is incorrect then the nearest neighbor used scaling instead that image quality is worse but faster. The recommended default is to set the filter to 'true' as the cost of the two-line filter is typically minimal and improves the image quality is significant. Bring back The Bethamp! New scaled bitmap or source bitmap is not scalded if needed. Exceptions java.lang.IllegalArgumentException if width is <= 0, or height is <= 0 fun eraseColor(c: Int): Unit Fills the bitmap's pixels with the specified Color. Java.lang.IllegalStateException exceptions if bitmap cannot be mutated. Fun eraseColor (Color: Long): Bitmap pixel fill unit is marked with ColorLong. Long Color Parameters: Paint to fill as packed by color class. Exception Java.lang.IllegalStateException if bitmap cannot be changed. java.lang.IllegalArgumentException if the color space encrypted in ColorLong is invalid or unknown. Fun extractAlpha(): Bitmap! Returns a new bitmap that captures original alpha values. This may be drawn with Canvas.drawBitmap(), where the color(s) will be taken from the color that is transferred to the draw call. Bring back The Bethamp! The new bitmap includes the alpha channel of the original Bitmap. Fun extractAlpha (Color: Color!, offsetXY: IntArray!): Bitmap! Returns a new bitmap that captures original alpha values. These values may be affected by the optional color parameter, which can contain your alpha, and may also include MaskFilter which can change the actual dimensions of the bitmap (for example a maskfilter blur may be a great result of bitmaps). If offsetXY is not empty, it returns the reverted bitmap compensation value to reasonably align with the original. For example, if the color contains a radius 2, then offsetXY[] will contain -2, -2, so that the offset draw of the alpha bitmap by (-2, -2) and then the original plot leads to visual blurring with the original. The initial density of the returned bitmap is the same as the original density. Color parameters: Optional color is used to change alpha values in resulting bitmaps. Pass empty for default behavior. OffsetXY IntArray!: An optional array that returns X (Index 0) and Y (Indicator 1) offsets the requirements for the bitmap position back so that it visually lines up with the original. Bring back The Bethamp! The new bitmap contains (optionally modified by color) alpha channel of the original Bitmap. This may be drawn with Canvas.drawBitmap(), where the color(s) will be taken from the color that is transferred to the draw call. Fun getByteCount(): Int returns the minimum number of bytes that can be used to store pixels of this bitmap. From android.os.Build.VERSION_CODES#KITKAT, the result of this method can no longer be used to determine the memory usage of a bitmap. See. Fun getColor(x: Int, y: Int): Returns the color color at the specified location. Throws an exception if x or y are out of limit (negative or >= in width or height, respectively). Parameters x Int: Coordinates x (0...width-1) pixels to return y Int: coordinates y (0...height-1) pixels for color return in specified coordinates This value cannot be depleted. Exceptions java.lang.IllegalArgumentException if x, y exceed the bitmap's bounds java.lang.IllegalStateException if the bitmap's config is Config#HARDWARE fun getColorSpace(): ColorSpace? Returns the color space associated with this bitmap. If the color space is unknown, this empty method returns. Fun getConfig(): Bitmap.Config! If the internal bitmap configuration is in one of the generic formats, restore that configuration, otherwise return the null. fun getGenerationId(): Int Returns the generation ID of this bitmap. The generation ID changes whenever bitmap is modified. It can be used as an efficient way to check whether a bitmap has changed. Return Int The current generation ID for this bitmap. Fun getHeight(): Int Back Height bitmap Fun getNinePatchChunk(): ByteArray! Returns an optional array of private data used by the UI system for some bitmaps. It is not intended to be recalled by applications. Fun getPixel(x: Int, y: Int): Int returns the color at the specified location. Throws an exception if x or y are out of limit (negative or >= in width or height, respectively). The back color is a non-premultiplied ARGB value in ColorSpace.Named#SRGB color space. Parameters x Int: Coordinates x (0...Width-1) Pixels for Return y Int: Coordinates y (0...Height-1) Pixels to return inte color argb in specified coordinates java.lang.IllegalArgumentException exception if x,y over the boundaries of bitmap Java.lang.IllegalStateException if the bitmap configuration is configured #سخت fun getPixels software (pixels: IntArray!, Offset: Int, Step: Int, x: Int, y: Int, Width: Int, Height: Int): Return unit in pixels[] a copy of data in bitmap. Each value of a packed int represents a color. The step parameter allows the caller to slot in the array of pixels returned between rows. For normal packed results, just pass the width for the step value. Returned colors are non-premultiplied ARGB values in ColorSpace.Named#SRGB colorspace. IntArray Pixel Parameters!: An array to get bitmap colors offset int: the first indicator that writes to pixels[] Step Int: The number of inputs in pixels[] to skip between rows (must >= the width of the bitmap). It can be negative. x Int: First pixel x coordinates for reading from bitmap y Int: first pixel y coordinates for reading from bitmap width Int: number of pixels to read from each row height Int: number of rows to read exceptions java.lang.IllegalArgumentException if x, y, width, height over bitmap boundaries, or if abs (step) width <. java.lang.ArrayIndexOutOfBoundsException if the pixel array is too small to get the specified number of pixels. java.lang.IllegalStateException if the bitmap's config is Config#HARDWARE fun getRowBytes(): Int Return the number of bytes between rows in the bitmap's pixels. Note that this refers to pixels as stored natively by bitmap. If you call getPixels() or setPixels(), then pixels are treated uniformly as 32bit values, packed according to the color class. From android.os.Build.VERSION_CODES#KITKAT, this method should not be used to calculate the use of bitmap memory. Instead, see getAllocationByteCount(). Return int number of bytes between rows of native bitmap pixels. fun getScaledHeight(targetDensity: Int): Int Convenience method that returns the height of this bitmap divided by the density scale factor. The height of the multiplied bitmap in the ratio of the target density to the density parameters of the source bitmap returns the target bitmap of the int density: the target canvas density returns the bitmap. Return int scaled height of this bitmap, according to the density scale factor. fun getScaledWidth(targetDensity: Int): Int Convenience method that returns the width of this bitmap divided by the density scale factor. The width of the multiplied bitmap in the ratio of the target density to the density parameters of the source bitmap returns the target bitmap of the int density: the target canvas density returns the bitmap. Return int scaled width of this bitmap, according to the density scale factor. Fun getWidth(): Int Back Bitmap's fun hasAlpha(): Boolean returns just if the bitmap configuration supports per alpha pixel, and if the pixel may contain non-opaque alpha values. For some configurations, this Always incorrect (as RGB_565), since they do not support any alpha pixels. However, for the configurations they do, Bitmap may be flagged to make it known that all its pixels are opaque. In this case, hasAlpha() also goes back incorrectly. If a configuration like ARGB_8888 not so flagged, it will return to the right default. Fun hasMipMap(): Boolean shows whether the rendering responsible for drawing this bitmap should attempt to use mipmaps when this bitmap is drawn scaled down. If you know that you want to draw this bitmap at less than 50% of its original size, you may be able to achieve higher quality this property is the only offer that can be ignored by rendering. It is not guaranteed to have any effect. Boolean's return is true if rendering should attempt to use mipmaps, false otherwise also fun isMutable(): Boolean returns correctly if bitmap is marked as mutant (as can be) fun isPremultiplied(): Boolean shows whether the pixels stored in these bitmaps are pre-multiplied. When a pixel is pre-multiplied, the RGB components are multiplied in the alpha component. For example, if the original color is a 50% transparent red (128, 255, 0, 0), the pre-multiplied form (128, 128, 0, 0) is. This method always returns incorrectly if getConfig() Bitmap.Config#RGB_565. The return value is not defined if getConfig() Bitmap.Config#ALPHA_8. This method only returns correctly if hasAlpha() returns properly. A bitmap without an alpha channel can be used both as a pre-multiplication and as a bitmap other than pre-multiplication. Only pre-multiplied bits may be drawn by the visibility system or canvas. If a bitmap other than pre-multiplied with an alpha channel is drawn to a canvas, anException run time will be thrown. Boolean returns just if the underlying pixels have been pre-multiplied, false otherwise fun isRecycled(): Boolean returns just if this bitmap is recycled. If so, then it's an error trying to access its pixels, and the bitmap doesn't draw. Boolean's return is true if the bitmap has been recycling the fun prepareToDraw(): the unit makes the bitmap-related cache that is used to draw it. Starting android.os.Build.VERSION_CODES#N, this call will start uploading asynchronously on the GPU in RenderThread, if Bitmap is not already uploaded. With hardware acceleration, Bitmaps must be uploaded to the GPU to render. This is done by default the first time a Bitmap is drawn, but this process can take several milliseconds depending on the size of the bitmap. Every time a Bitmap is modified and re-drawn, it needs to be re-uploaded. Calling this method can already save time in the first frame used. For example, it is recommended to call this on an image decryption worker thread when a decrypted bitmap is being displayed. It is recommended that any changes be made to Bitmap before calling this method, so copied, uploaded without re-uploading may be used again. In And below, to clean up the cleanup This call tries to ensure that the pixels are decrypted. Fun reconfiguration (Width: Int, Height: Int, Configuration: Bitmap.Config!): bitmap correction unit to specified width, height, and configuration, without affecting the underlying allocation of bitmap support. Pixel Bitmap data has not been re-initialized for the new configuration. This method can be used to prevent the allocation of a new bitmap, instead reusing the existing bitmap allocation for a new configuration of equal to or less size. If bitmap allocation is not large enough to support the new configuration, illegalArgumentException will be launched and bitmap will not be modified. The getByteCount() result will reflect the new configuration, while GetAllocationByteCount() will reflect that of the initial configuration. Note: This may change this hasAlpha() result. When converting to 565, the new bitmap will always be considered opaque. When converting from 565, the new bitmap will be considered opaque, and the value set by setPremultiplied() will be respected. Warning: This method should not be invoked on a bitmap currently in use by the calling viewing system, boom, or AndroidBitmap NDK API. This guarantees on how the underlying pixel buffer does not rebuild into the new configuration, just that reuse allocation. In addition, the display system does not account for bitmap properties changing during use, as such while connected to drawables. To safely ensure that another Bitmap is not being used by the View system, it is necessary to wait for a lottery pass to occur after the unrealization()'ing any view that had previously drawn Bitmap in the last pass of the draw due to caching hardware acceleration from the lottery commands. For example, here's how to do this for ImageView: ImageView myImageView = ...; myBitmap final bitmap = ...; myImageView.setImageDrawable(null); myImageView.post(new Runnable() { public void run() { // myBitmap is now no longer in use by the ImageView // and can be safely reconfigured. myBitmap.reconfigure(...); }); also #setWidth (int) #setHeight(int) #setConfig(configuration) Fun recycling(): free unit native object associated with this bitmap, and turn on reference to pixel data. Bitmap is marked as dead, meaning it's a throwing exception if it's called getPixels() or setPixels(), and nothing draws. When there is no mention of this bitmap anymore. Fun sameAs(other: Bitmap!): Boolean according to another bitmap, return just if it has the same dimensions, configuration, and pixel data as this bitmap. If any of it is different, the return is incorrect. If the other is invalid, incorrectly return it. funny ColorSpace): Unit Modifies the bitmap to have the specified ColorSpace, without affecting the underlying allocation backing the bitmap. This will affect how the color frame is interpreted per pixel. A bitmap with Config#ALPHA_8 never have color space, because a color space doesn't affect the alpha channel. Other configurations should always have a non-empty color space. ColorSpace Parameters: This value cannot be depleted to assign bitmaps. Fun setHasAlpha (hasAlpha: Boolean): Units tell Bitmap if all pixels are known to be opaque (incorrect) or if some pixels may contain non-opaque alpha values (true). Note, for some configurations (as RGB_565) this call is ignored because it does not support alpha values of each pixel. This is meant as a hint of painting, as in some cases a bitmap that is known to be opaque can make a painting case faster than one that may be opaque per pixel alpha value. Fun setHasMipMap(hasMipMap: Boolean): The unit setting a hint for the rendering responsible for drawing this bitmap indicates that it should attempt to use mipmaps when this bitmap is drawn scaled down. If you know that you want to draw this bitmap at less than 50% of its original size, you may be able to achieve higher quality by turning on this property. Note that if the rendering respects this mention may have to allocate additional memory to keep mipmap levels for this bitmap. The property is just one offer that can be ignored by rendering. It is not guaranteed to have any effect. Parameters hasMipMap Boolean: Indicates whether rendering should be an attempt to use setPixel's fun mipmaps (x: Int, y: Int, Color: Int): The color writing unit specified into bitmap (assuming it is mutant) in the coordinates x,y. Color should be a non-premultiplied ARGB value in ColorSpace.Named#SRGB color space. Parameters x Int: Coordinates x pixels instead of (0...width-1) y Int: coordinates y pixels instead (0...height-1) Color Int: ARGB color to write to bitmap java.lang.IllegalStateException exceptions if bitmap is not mutant java.lang.illegalArgumentExEx if x, y are out of bitmap boundaries. Fun setPixels (pixels: IntArray!, offset: Int, Step: Int, x: Int, y: Int, Width: Int, Height: Int): Pixel replacement unit in bitmap with color in array. Each element in the packed int array represents a non-premultiplied ARGB color in ColorSpace.Named#SRGB color space. IntArray Pixel Parameters!: Color to write to bitmap Offset Int: The first color indicator to read from pixels[] Step Int: Color count in pixels[] to jump between rows. Normally this value will be the same as the width of bitmap, but it can be larger (or negative). x Int: Coordinates the first x pixels to write to in bitmap. y Int: Coordinates the first pixel y to write to in bitmap. Int Width: Number of colors to copy from pixels [] per row Int height: number of rows to to the bitmap Exceptions java.lang.IllegalStateException if the bitmap is not mutable java.lang.IllegalArgumentException if x, y, width, height are outside of the bitmap's bounds. java.lang.ArrayIndexOutOfBoundsException if the pixel array is too small to get the specified number of pixels. Fun setPremultiplied (premultiplied: Boolean): Set units whether bitmap should treat your data as pre-multiplied. Bitmaps are always treated as pre-multiplied by the display system and canvas for performance reasons. Storing un-pre-multiplied data in a Bitmap (via setPixel, setPixels, or BitmapFactory.Options#inPremultiplied) can lead to incorrect composition if drawn by the framework. This method does not affect the behavior of a bitmap without an alpha channel, or if hasAlpha() returns incorrectly. Calling createBitmap or createScaledBitmap with a Bitmap source that has not been pre-multiplied may lead to anException run-off time, as those functions require a source drawing, which is not supported for un-multiplied bitmaps. static fun wrapHardwareBuffer(hardwareBuffer: HardwareBuffer, colorSpace: ColorSpace?): Bitmap? Create a hardware bitmap with the support of a HardwareBuffer. Flags using HardwareBuffer must be passed containing HardwareBuffer#USAGE_GPU_SAMPLED_IMAGE. Bitmap will hold a reference to the buffer so that callers can safely pack hardwareBuffer without affecting Bitmap. However HardwareBuffer should not be modified while Bitmap wrapped is accessing it. Doing so will lead to undefined behavior. Hardware ParametersBuffer HardwareBuffer: Buffer hardware to wrap. This value cannot be depleted. ColorSpace ColorSpace:: Bitmap color space. It should be colorSpace.Rgb color space. If it's empty, SRGB is assumed. This value may be empty. Bring back The Bethamp? Bitmap is wrapping the buffer, or empty if there was a problem with the bitmap. Java.lang.IllegalArgumentException exceptions if HardwareBuffer has an invalid use, or invalid color space. fun writeToParcel(p: Parcel!, flags: Int): Unit Write the bitmap and its pixels to the parcel. Bitmap can be restored from the package by calling CREATOR.createFromParcel(). If this bitmap is Config#HARDWARE, it may not be packed in a different pixel format (such as 565, 8888), but the content will be preserved to the best quality allowed by the ultimate static pixel format val CREATOR: Parcelable.Creator<Bitmap!> Parcelable.Creator<Bitmap!> > </Bitmap!>