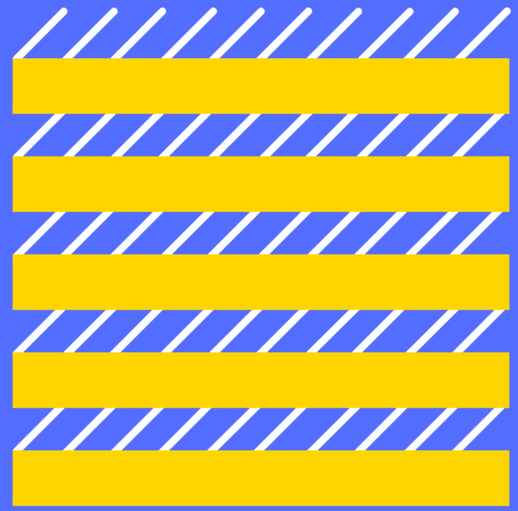
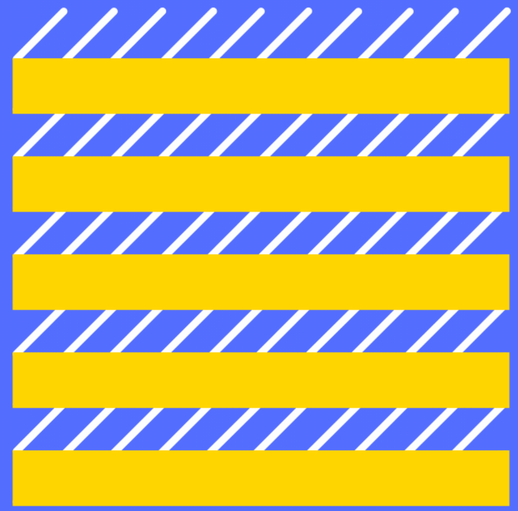
Android KTX



Jake Wharton

Android KTX



Jake Wharton

```kotlin
val userLayout: ViewGroup = findViewById(R.id.users)
for (index in 0 until userLayout.childCount) {
    val view = userLayout.getChildAt(index)
    // Do something with index and view…
}
```

```kotlin
fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

val userLayout: ViewGroup = findViewById(R.id.users)
for (index in 0 until userLayout.childCount) {
    val view = userLayout.getChildAt(index)
    // Do something with index and view…
}
```

```kotlin
fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

val userLayout: ViewGroup = findViewById(R.id.users)
for (index in 0 until userLayout.childCount) {
    val view = userLayout.getChildAt(index)
    // Do something with index and view…
}
```

```kotlin
fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

val userLayout: ViewGroup = findViewById(R.id.users)
for (index in 0 until userLayout.childCount) {
    val view = userLayout.getChildAt(index)
    // Do something with index and view…
}
```

```kotlin
fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

val userLayout: ViewGroup = findViewById(R.id.users)
userLayout.forEachIndexed { index, view ->
    // Do something with index and view…
}
```

```kotlin
fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

val userLayout: ViewGroup = findViewById(R.id.users)
userLayout.forEachIndexed { index, view ->
    // Do something with index and view…
}
```

```kotlin
fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

val userLayout: ViewGroup = findViewById(R.id.users)
userLayout.forEachIndexed { index, view ->
    // Do something with index and view…
}
```

```kotlin
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

val userLayout: ViewGroup = findViewById(R.id.users)
userLayout.forEachIndexed { index, view ->
    // Do something with index and view…
}
```

```kotlin
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

val userLayout: ViewGroup = findViewById(R.id.users)
userLayout.forEachIndexed { index, view ->
    // Do something with index and view…
}
```

```
// In an Activity, on API 23+…
val notifications = getSystemService(NotificationManager::class.java)
```

```kotlin
// In an Activity, on API 23+…
val notifications = getSystemService(NotificationManager::class.java)

// or all API levels…
val notifications = ContextCompat.getSystemService(this,
    NotificationManager::class.java)
```

```
inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)


// In an Activity, on API 23+…
val notifications = getSystemService(NotificationManager::class.java)

// or all API levels…
val notifications = ContextCompat.getSystemService(this,
    NotificationManager::class.java)
```

```
inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)


// In an Activity, on API 23+…
val notifications = getSystemService(NotificationManager::class.java)

// or all API levels…
val notifications = ContextCompat.getSystemService(this,
    NotificationManager::class.java)
```

```
inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)


// In an Activity, on API 23+…
val notifications = getSystemService(NotificationManager::class.java)

// or all API levels…
val notifications = ContextCompat.getSystemService(this,
    NotificationManager::class.java)
```

```
inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)


// In an Activity, on API 23+…
val notifications = getSystemService(NotificationManager::class.java)

// or all API levels…
val notifications = ContextCompat.getSystemService(this,
    NotificationManager::class.java)
```

```
inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)


// In an Activity…
val notifications = systemService<NotificationManager>()
```

```kotlin
inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)


// In an Activity…
val notifications = systemService<NotificationManager>()
```

```
avatarView.setPadding(
    10, avatarView.paddingTop, 10, avatarView.paddingBottom)
```

```kotlin
inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.setPadding(
    10, avatarView.paddingTop, 10, avatarView.paddingBottom)
```

```kotlin
inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.setPadding(
    10, avatarView.paddingTop, 10, avatarView.paddingBottom)
```

```kotlin
inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.setPadding(
    10, avatarView.paddingTop, 10, avatarView.paddingBottom)
```

```kotlin
inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.updatePadding(10, 10)
```

```kotlin
inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.updatePadding(10, 10)
```

```kotlin
inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.updatePadding(left = 10, right = 10)
```

```kotlin
inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.updatePadding(left = 10, right = 10)
```

```
val rect = avatarView.clipBounds
val left = rect.left
val top = rect.top
val right = rect.right
val bottom = rect.bottom
// Use left, top, right, bottom…
```

```
inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

val rect = avatarView.clipBounds
val left = rect.left
val top = rect.top
val right = rect.right
val bottom = rect.bottom
// Use left, top, right, bottom…
```

```
inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

val rect = avatarView.clipBounds
val left = rect.left
val top = rect.top
val right = rect.right
val bottom = rect.bottom
// Use left, top, right, bottom…
```

```kotlin
inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

val rect = avatarView.clipBounds
val left = rect.left
val top = rect.top
val right = rect.right
val bottom = rect.bottom
// Use left, top, right, bottom…
```

```kotlin
inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

val rect = avatarView.clipBounds
val left = rect.left
val top = rect.top
val right = rect.right
val bottom = rect.bottom
// Use left, top, right, bottom…
```

```kotlin
inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

val (left, top, right, bottom) = avatarView.clipBounds
// Use left, top, right, bottom…
```

```
inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

val (left, top, right) = avatarView.clipBounds
// Use left, top, right…
```

```
inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

val (left, _, right) = avatarView.clipBounds
// Use left, right…
```

```
inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

val (left, _, right) = avatarView.clipBounds
// Use left, right…
```

```
var onlyDigits = true
for (c in phoneNumber) {
    if (!c.isDigit()) {
        onlyDigits = false
        break
    }
}
```

```
val onlyDigits = phoneNumber.all { it.isDigit() }
```

```
val onlyDigits = TextUtils.isDigitsOnly(phoneNumber)
```

```
inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = TextUtils.isDigitsOnly(phoneNumber)
```

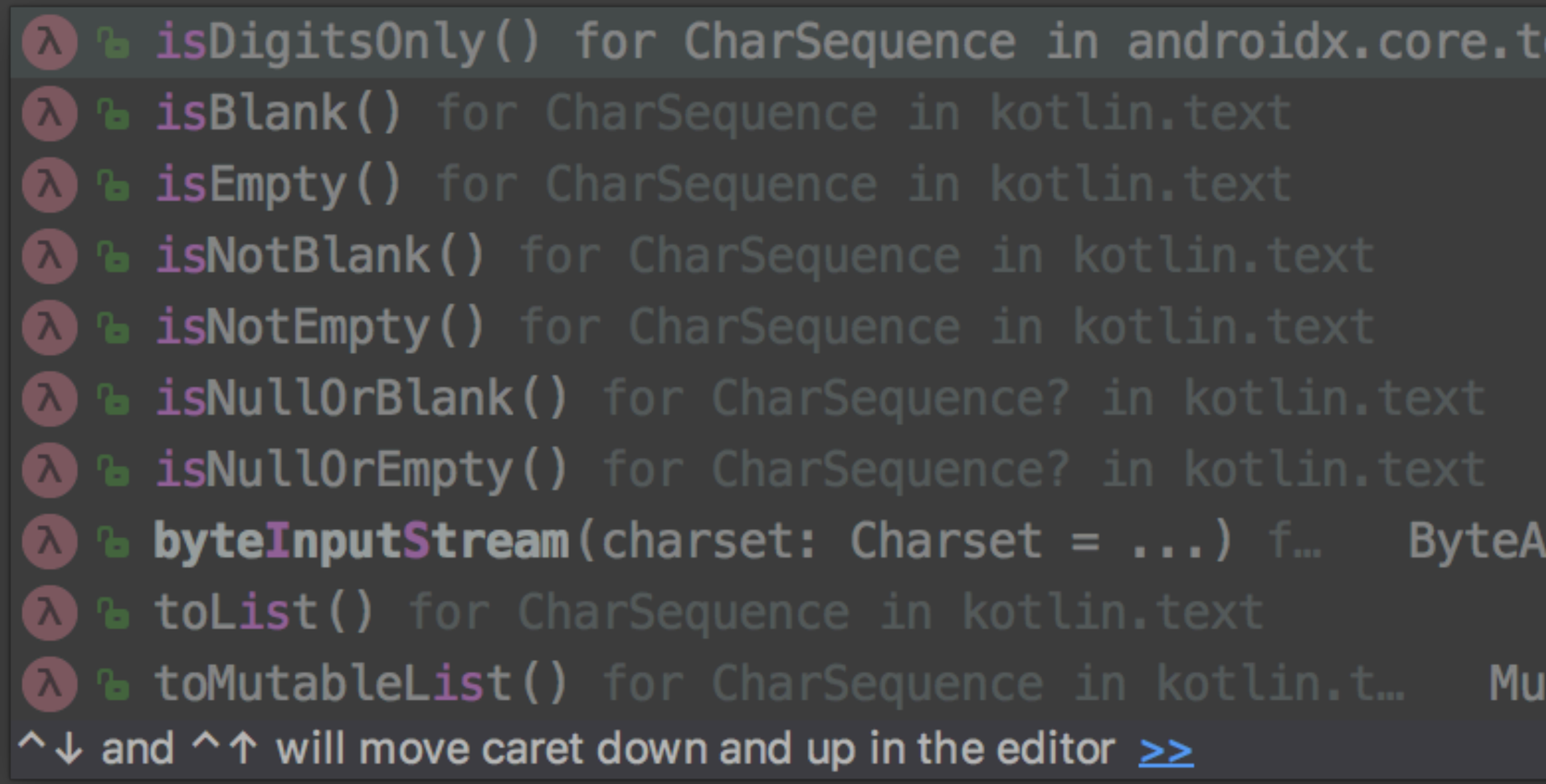
```kotlin
inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.isDigitsOnly()
```

```
inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.|
```

```
inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.|
```

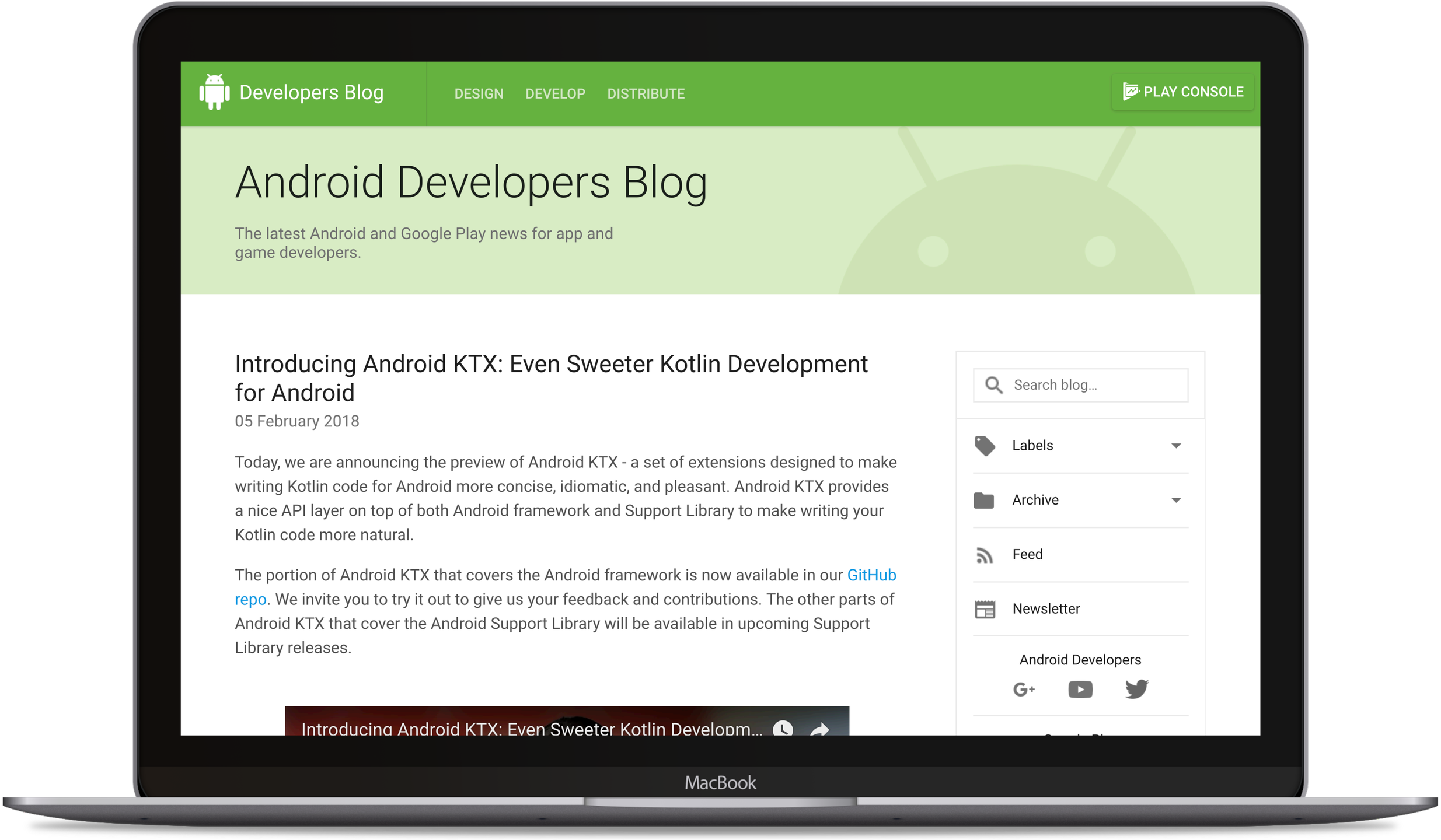
```
inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.is
```

```
inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.is
```



```kotlin
inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.isDigitsOnly()
```

```kotlin
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)

inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)
```


Developers Blog
DESIGN
DEVELOP
DISTRIBUTE
PLAY CONSOLE
Android Developers Blog
The latest Android and Google Play news for app and game developers.
Introducing Android KTX: Even Sweeter Kotlin Development for Android
05 February 2018
Today, we are announcing the preview of Android KTX - a set of extensions designed to make writing Kotlin code for Android more concise, idiomatic, and pleasant. Android KTX provides a nice API layer on top of both Android framework and Support Library to make writing your Kotlin code more natural.
The portion of Android KTX that covers the Android framework is now available in our GitHub repo. We invite you to try it out to give us your feedback and contributions. The other parts of Android KTX that cover the Android Support Library will be available in upcoming Support Library releases.
Introducing Android KTX: Even Sweeter Kotlin Developm...
Search blog...
Labels
Archive
Feed
Newsletter
Android Developers
MacBook

Android KTX

core-ktx

Android KTX

core-ktx → Android framework

Android KTX

core-ktx → support-compat → Android framework

Android KTX

core-ktx → core → Android framework

Android KTX

core-ktx → core → Android framework

fragment-ktx → fragment

palette-ktx → palette

collection-ktx → collection

lifecycle-reactivestreams-ktx → lifecycle-reactivestreams

sqlite-ktx → sqlite

navigation-*-ktx → navigation-*

work-runtime-ktx → work-runtime

Android KTX

core-ktx → core → Android framework

fragment-ktx → fragment

palette-ktx → palette

collection-ktx → collection

lifecycle-reactivestreams-ktx → lifecycle-reactivestreams

sqlite-ktx → sqlite

navigation-*-ktx → navigation-*

work-runtime-ktx → work-runtime

```kotlin
@RequiresApi(26)
operator fun Color.plus(c: Color): Color {
    val s = if (colorSpace != c.colorSpace) c.convert(colorSpace) else c

    val src = s.components
    val dst = components

    var sa = s.alpha()
    // Destination alpha pre-composited
    var da = alpha() * (1.0f - sa)

    // Index of the alpha component
    val ai = componentCount - 1

    // Final alpha: src_alpha + dst_alpha * (1 - src_alpha)
    dst[ai] = sa + da

    // Divide by final alpha to return non pre-multiplied color
    if (dst[ai] > 0) {
        sa /= dst[ai]
        da /= dst[ai]
    }

    // Composite non-alpha components
    for (i in 0 until ai) {
        dst[i] = src[i] * sa + dst[i] * da
    }

    return Color.valueOf(dst, colorSpace)
}
```

```kotlin
@RequiresApi(26)
operator fun Color.plus(c: Color): Color {
    val s = if (colorSpace != c.colorSpace) c.convert(colorSpace) else c

    val src = s.components
    val dst = components

    var sa = s.alpha()
    // Destination alpha pre-composited
    var da = alpha() * (1.0f - sa)

    // Index of the alpha component
    val ai = componentCount - 1

    // Final alpha: src_alpha + dst_alpha * (1 - src_alpha)
    dst[ai] = sa + da

    // Divide by final alpha to return non pre-multiplied color
    if (dst[ai] > 0) {
        sa /= dst[ai]
        da /= dst[ai]
    }

    // Composite non-alpha components
    for (i in 0 until ai) {
        dst[i] = src[i] * sa + dst[i] * da
    }

    return Color.valueOf(dst, colorSpace)
}
```

```kotlin
@RequiresApi(26)
operator fun Color.plus(c: Color): Color {
    val s = if (colorSpace != c.colorSpace) c.convert(colorSpace) else c

    val src = s.components
    val dst = components

    var sa = s.alpha()
    // Destination alpha pre-composited
    var da = alpha() * (1.0f - sa)

    // Index of the alpha component
    val ai = componentCount - 1

    // Final alpha: src_alpha + dst_alpha * (1 - src_alpha)
    dst[ai] = sa + da

    // Divide by final alpha to return non pre-multiplied color
    if (dst[ai] > 0) {
        sa /= dst[ai]
        da /= dst[ai]
    }

    // Composite non-alpha components
    for (i in 0 until ai) {
        dst[i] = src[i] * sa + dst[i] * da
    }

    return Color.valueOf(dst, colorSpace)
}
```

```
public final class ColorUtils {
}
```

```
public final class ColorUtils {
    @ColorInt
    public static int compositeColors(
            @ColorInt int foreground, @ColorInt int background) {
        // …
    }
}
```

```
public final class ColorUtils {
    @ColorInt
    public static int compositeColors(
            @ColorInt int foreground, @ColorInt int background) {
        // …
    }

    @RequiresApi(26)
    public static Color compositeColors(
            Color foreground, Color background) {
        // …
    }
}
```

```kotlin
@RequiresApi(26)
operator fun Color.plus(c: Color): Color {
    val s = if (colorSpace != c.colorSpace) c.convert(colorSpace) else c

    val src = s.components
    val dst = components

    var sa = s.alpha()
    // Destination alpha pre-composited
    var da = alpha() * (1.0f - sa)

    // Index of the alpha component
    val ai = componentCount - 1

    // Final alpha: src_alpha + dst_alpha * (1 - src_alpha)
    dst[ai] = sa + da

    // Divide by final alpha to return non pre-multiplied color
    if (dst[ai] > 0) {
        sa /= dst[ai]
        da /= dst[ai]
    }

    // Composite non-alpha components
    for (i in 0 until ai) {
        dst[i] = src[i] * sa + dst[i] * da
    }

    return Color.valueOf(dst, colorSpace)
}
```

```kotlin
@RequiresApi(26)
inline operator fun Color.plus(c: Color): Color =
    ColorUtils.compositeColors(c, this)
```

```kotlin
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)

inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)
```

```kotlin
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)

inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)
```

KTX Principles

KTX Principles

Adapt existing functionality and redirect features upstream

```kotlin
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)

inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)
```

```kotlin
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)

inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)
```

```kotlin
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)

inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)
```

```kotlin
operator fun ViewGroup.iterator() = object : MutableIterator<View> {
    private var index = 0
    override fun hasNext() = index < childCount
    override fun next() =
        getChildAt(index++) ?: throw IndexOutOfBoundsException()
    override fun remove() = removeViewAt(--index)
}
```

```
operator fun ViewGroup.iterator() = object : MutableIterator<View> {
    private var index = 0
    override fun hasNext() = index < childCount
    override fun next() =
        getChildAt(index++) ?: throw IndexOutOfBoundsException()
    override fun remove() = removeViewAt(--index)
}

for (view in userLayout) {
    // Do something with view…
}
```

```
operator fun ViewGroup.iterator() = object : MutableIterator<View> {
    private var index = 0
    override fun hasNext() = index < childCount
    override fun next() =
        getChildAt(index++) ?: throw IndexOutOfBoundsException()
    override fun remove() = removeViewAt(--index)
}

for (view in userLayout) {
    // Do something with view…
}
```

```
operator fun ViewGroup.iterator() = object : MutableIterator<View> {
    private var index = 0
    override fun hasNext() = index < childCount
    override fun next() =
        getChildAt(index++) ?: throw IndexOutOfBoundsException()
    override fun remove() = removeViewAt(--index)
}

for (view in userLayout) {
    // Do something with view…
}
```

KTX Principles

Adapt existing functionality and redirect features upstream

KTX Principles

- Adapt existing functionality and redirect features upstream
- Default to inline unless code size or allocation is prohibitive

```kotlin
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
    for (index in 0 until childCount) {
        action(index, getChildAt(index))
    }
}

val userLayout: ViewGroup = findViewById(R.id.users)
userLayout.forEachIndexed { index, view ->
    // Do something with index and view…
}
```

```
inline fun <reified T> Context.systemService() =
    ContextCompat.getSystemService(this, T::class.java)


// In an Activity…
val notifications = systemService<NotificationManager>()
```

```kotlin
inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.updatePadding(left = 10, right = 10)
```

```
inline operator fun Rect.component1() = left
inline operator fun Rect.component2() = top
inline operator fun Rect.component3() = right
inline operator fun Rect.component4() = bottom

val (left, _, right) = avatarView.clipBounds
// Use left, right…
```

```kotlin
inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.isDigitsOnly()
```

```
operator fun ViewGroup.iterator() = object : MutableIterator<View> {
    private var index = 0
    override fun hasNext() = index < childCount
    override fun next() =
        getChildAt(index++) ?: throw IndexOutOfBoundsException()
    override fun remove() = removeViewAt(--index)
}

for (view in userLayout) {
    // Do something with view…
}
```

KTX Principles

- Adapt existing functionality and redirect features upstream
- Default to inline unless code size or allocation is prohibitive

KTX Principles

- Adapt existing functionality and redirect features upstream
- Default to inline unless code size or allocation is prohibitive
- Leverage features unique to Kotlin

```
view.setOnClickListener {
    // React to click…
}
```

```
fun View.click(listener: (View) -> Unit) {
    setOnClickListener(listener)
}

view.setOnClickListener {
    // React to click…
}
```

```kotlin
fun View.click(listener: (View) -> Unit) {
    setOnClickListener(listener)
}

view.setOnClickListener {
    // React to click…
}
```

```
fun View.click(listener: (View) -> Unit) {
    setOnClickListener(listener)
}

view.click {
    // React to click…
}
```

```
view.setOnClickListener {
    // React to click…
}

view.click {
    // React to click…
}
```

KTX Principles

- Adapt existing functionality and redirect features upstream
- Default to inline unless code size or allocation is prohibitive
- Leverage features unique to Kotlin

KTX Principles


Adapt existing functionality and redirect features upstream


Default to inline unless code size or allocation is prohibitive


Leverage features unique to Kotlin


Code golf APIs to be as short as possible

```
if (Build.VERSION.SDK_INT >= 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}
```

```
inline fun onApi(level: Int, body: () -> Unit) {
    if (Build.VERSION.SDK_INT >= level) {
        body()
    }
}

if (Build.VERSION.SDK_INT > 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}
```

```
inline fun onApi(level: Int, body: () -> Unit) {
    if (Build.VERSION.SDK_INT >= level) {
        body()
    }
}

if (Build.VERSION.SDK_INT > 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}
```

```
inline fun onApi(level: Int, body: () -> Unit) {
    if (Build.VERSION.SDK_INT >= level) {
        body()
    }
}

onApi(19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}
```

```
onApi(19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}
```

```
if (Build.VERSION.SDK_INT >= 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}

onApi(19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}
```

```
if (SDK_INT >= 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}

onApi(19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}
```

```
if (SDK_INT >= 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}

onApi(19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}
```

```
if (SDK_INT >= 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
} else {
    // Something...
}

onApi(19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}
```

```
if (SDK_INT >= 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
} else {
    // Something...
}

onApi(19) {
    TransitionManager.beginDelayedTransition(viewGroup)
}
```

```
if (SDK_INT >= 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
} else {
    // Something...
}

onApi(19, {
    TransitionManager.beginDelayedTransition(viewGroup)
}, {
    // Something...
})
```

```
if (SDK_INT >= 28) {
    // Fancy new thing...
} else if (SDK_INT >= 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
} else {
    // Something...
}

onApi(19, {
    TransitionManager.beginDelayedTransition(viewGroup)
}, {
    // Something...
})
```

```
if (SDK_INT >= 28) {
    // Fancy new thing...
} else if (SDK_INT >= 19) {
    TransitionManager.beginDelayedTransition(viewGroup)
} else {
    // Something...
}

onApi(19, {
    TransitionManager.beginDelayedTransition(viewGroup)
}, {
    // Something...
})
```

KTX Principles


Adapt existing functionality and redirect features upstream


Default to inline unless code size or allocation is prohibitive


Leverage features unique to Kotlin


Code golf APIs to be as short as possible

KTX Principles


Adapt existing functionality and redirect features upstream


Default to inline unless code size or allocation is prohibitive


Leverage features unique to Kotlin


Code golf APIs to be as short as possible


Optimize for a single and/or specific use case

Android KTX

core-ktx → core → Android framework

fragment-ktx → fragment

palette-ktx → palette

collection-ktx → collection

lifecycle-reactivestreams-ktx → lifecycle-reactivestreams

sqlite-ktx → sqlite

navigation-*-ktx → navigation-*

work-runtime-ktx → work-runtime

Android KTX

core-ktx → core → Android framework

fragment-ktx → fragment

palette-ktx → palette

collection-ktx → collection

lifecycle-reactivestreams-ktx → lifecycle-reactivestreams

sqlite-ktx → sqlite

navigation-*-ktx → navigation-*

work-runtime-ktx → work-runtime

```
supportFragmentManager.beginTransaction()
    .replace(android.R.id.content, userFragment)
    .commit()
```

```
inline fun FragmentManager.transaction(
    body: FragmentTransaction.() -> Unit
) {
    val transaction = beginTransaction()
    transaction.body()
    transaction.commit()
}

supportFragmentManager.beginTransaction()
    .replace(android.R.id.content, userFragment)
    .commit()
```

```
inline fun FragmentManager.transaction(
    body: FragmentTransaction.() -> Unit
) {
    val transaction = beginTransaction()
    transaction.body()
    transaction.commit()
}

supportFragmentManager.beginTransaction()
    .replace(android.R.id.content, userFragment)
    .commit()
```

```
inline fun FragmentManager.transaction(
    body: FragmentTransaction.() -> Unit
) {
    val transaction = beginTransaction()
    transaction.body()
    transaction.commit()
}

supportFragmentManager.transaction {
    replace(android.R.id.content, userFragment)
}
```

```kotlin
inline fun FragmentManager.transaction(
    allowStateLoss: Boolean = false
    body: FragmentTransaction.() -> Unit
) {
    val transaction = beginTransaction()
    transaction.body()
    if (allowStateLoss) transaction.commitAllowingStateLoss()
    else transaction.commit()
}

supportFragmentManager.transaction {
    replace(android.R.id.content, userFragment)
}
```

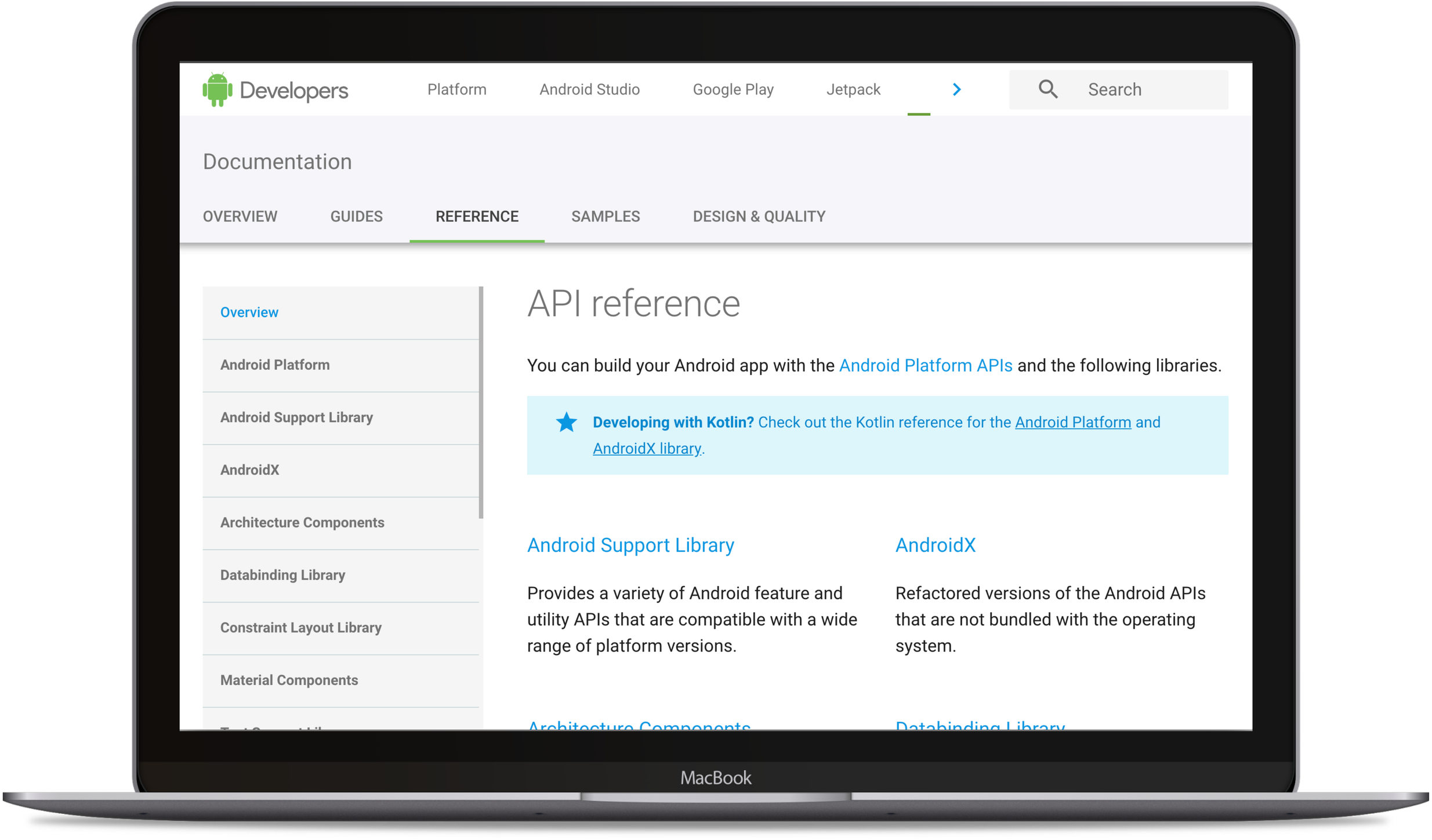
```
inline fun FragmentManager.transaction(
    allowStateLoss: Boolean = false
    body: FragmentTransaction.() -> Unit
) {
    val transaction = beginTransaction()
    transaction.body()
    if (allowStateLoss) transaction.commitAllowingStateLoss()
    else transaction.commit()
}

supportFragmentManager.transaction {
    replace(android.R.id.content, userFragment)
}
```

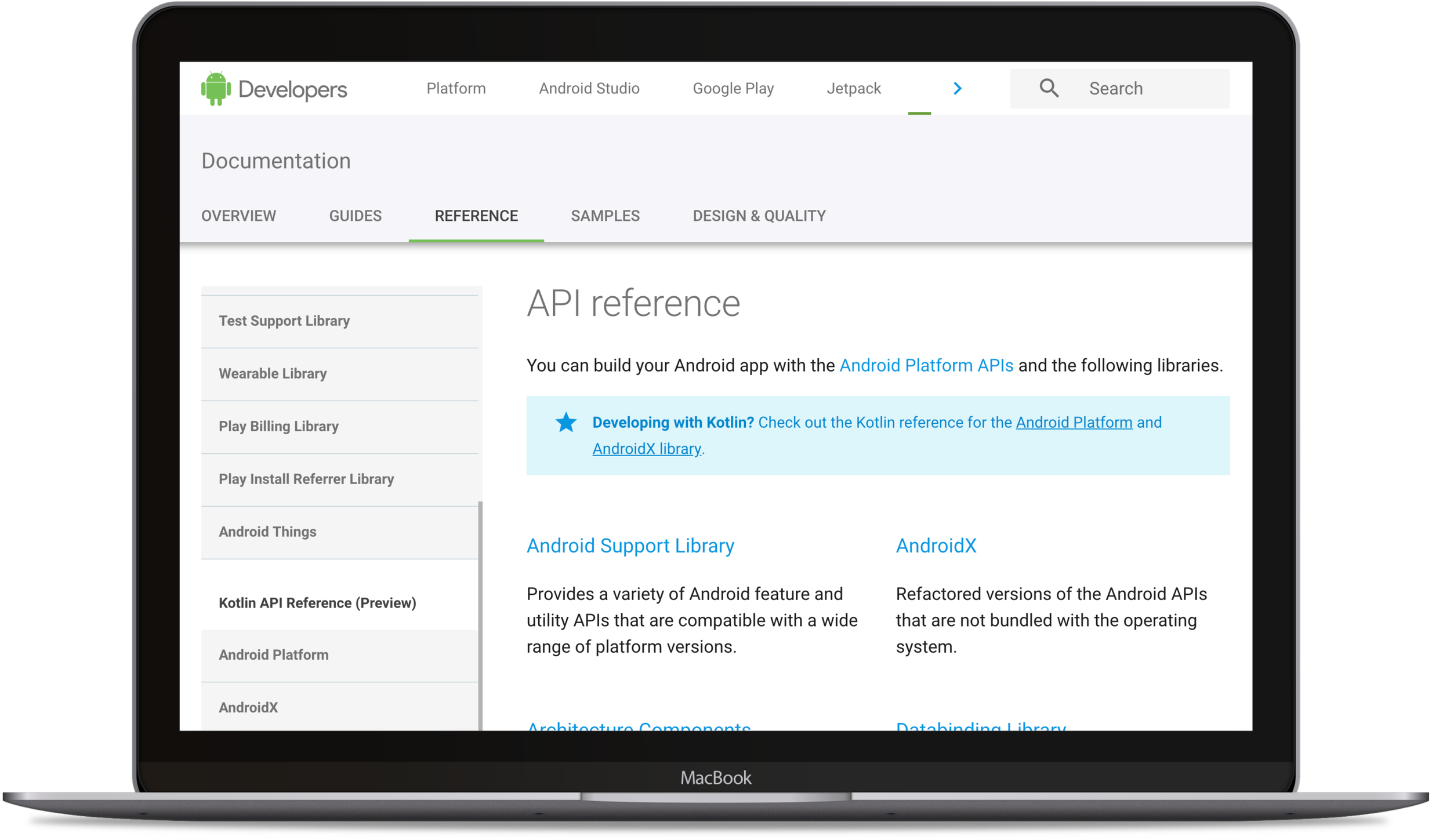
```
inline fun FragmentManager.transaction(
    allowStateLoss: Boolean = false,
    body: FragmentTransaction.() -> Unit
) {
    val transaction = beginTransaction()
    transaction.body()
    if (allowStateLoss) transaction.commitAllowingStateLoss()
    else transaction.commit()
}

supportFragmentManager.transaction(allowStateLoss = true) {
    replace(android.R.id.content, userFragment)
}
```

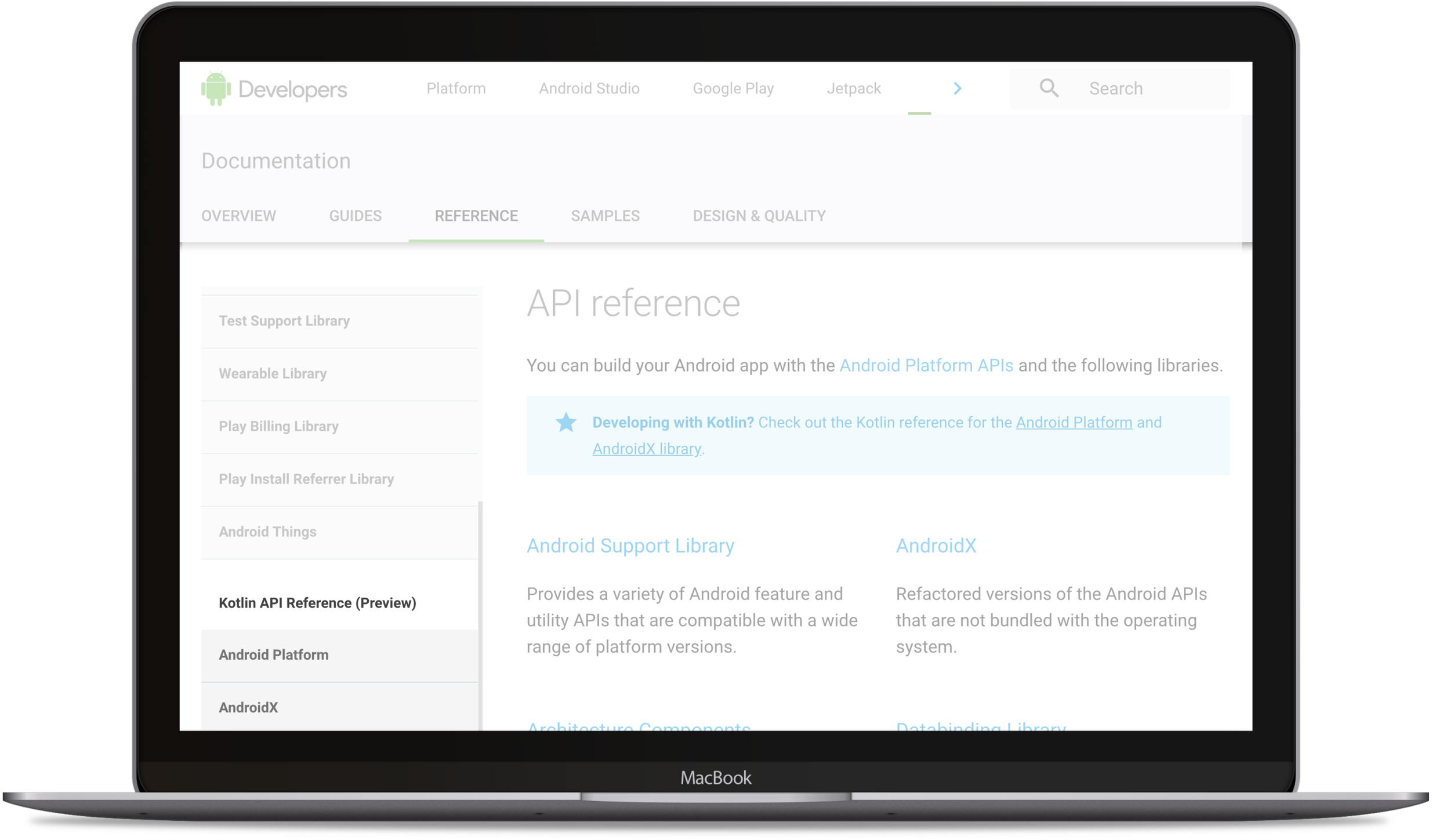
```
L7    ALOAD 3
      INVOKEVIRTUAL androidx/fragment/app/FragmentManager
          beginTransaction()
          Landroidx/fragment/app/FragmentTransaction;
      ASTORE 7

L8    ALOAD 7
      LDC 16908290
      ALOAD 2
      INVOKEVIRTUAL androidx/fragment/app/FragmentTransaction
          replace( I Landroidx/fragment/app/Fragment; )
          Landroidx/fragment/app/FragmentTransaction;
      POP

L9    ALOAD 7
      INVOKEVIRTUAL androidx/fragment/app/FragmentTransaction
          commitAllowingStateLoss()
          I
      POP
```

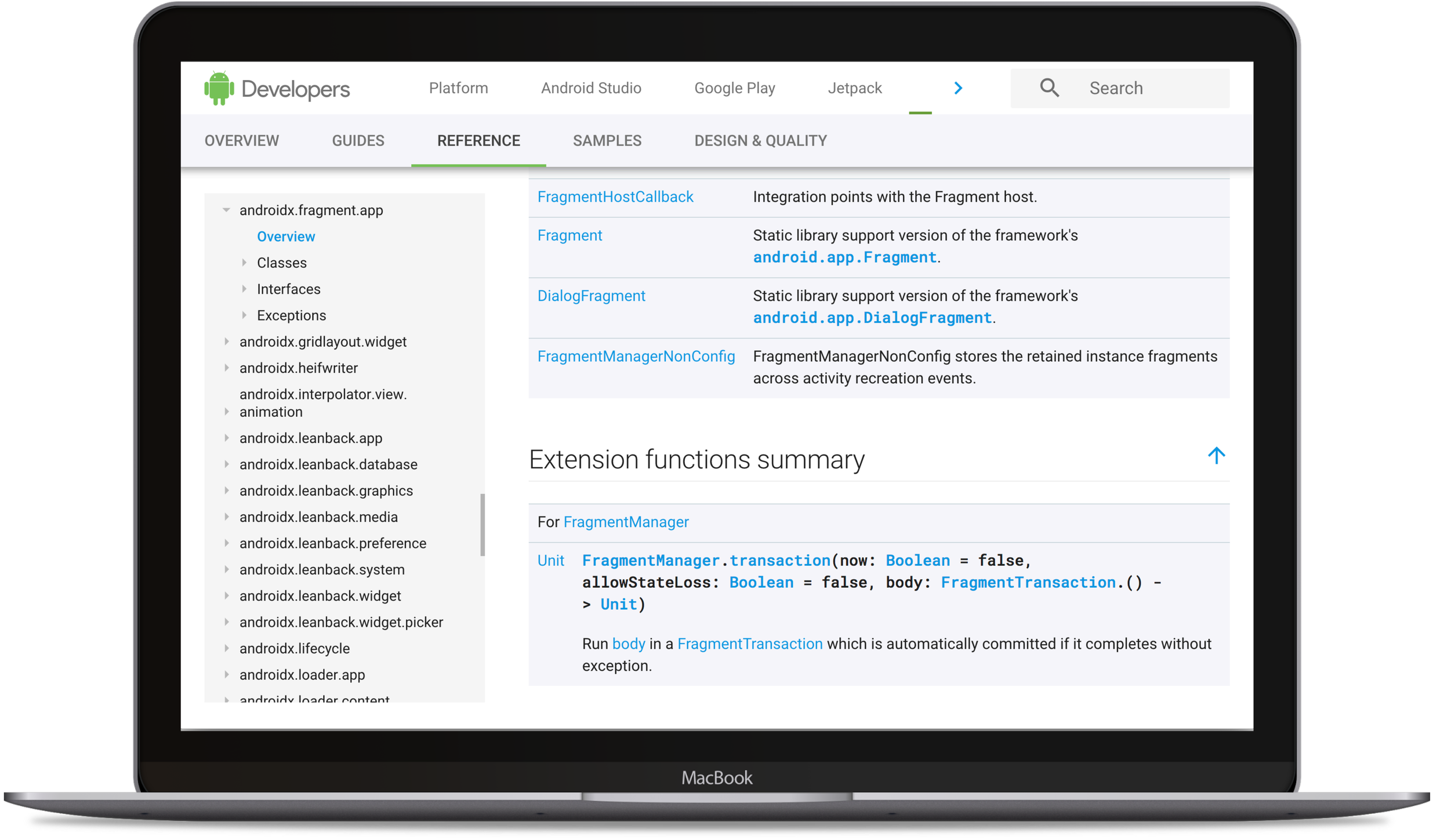
```
inline fun FragmentManager.transaction(
    allowStateLoss: Boolean = false,
    body: FragmentTransaction.() -> Unit
) {
    val transaction = beginTransaction()
    transaction.body()
    if (allowStateLoss) transaction.commitAllowingStateLoss()
    else transaction.commit()
}

supportFragmentManager.transaction(allowStateLoss = true) {
    replace(android.R.id.content, userFragment)
}
```

```
inline fun FragmentManager.transaction(
    now: Boolean = false,
    allowStateLoss: Boolean = false,
    body: FragmentTransaction.() -> Unit
) {
    val transaction = beginTransaction()
    transaction.body()
    if (now) {
        if (allowStateLoss) transaction.commitNowAllowingStateLoss()
        else transaction.commitNow()
    } else {
        if (allowStateLoss) transaction.commitAllowingStateLoss()
        else transaction.commit()
    }
}
```


Developers
Platform
Android Studio
Google Play
Jetpack
Search
Documentation
OVERVIEW
GUIDES
REFERENCE
SAMPLES
DESIGN & QUALITY
Overview
Android Platform
Android Support Library
AndroidX
Architecture Components
Databinding Library
Constraint Layout Library
Material Components
API reference
You can build your Android app with the Android Platform APIs and the following libraries.
Developing with Kotlin? Check out the Kotlin reference for the Android Platform and AndroidX library.
Android Support Library
Provides a variety of Android feature and utility APIs that are compatible with a wide range of platform versions.
AndroidX
Refactored versions of the Android APIs that are not bundled with the operating system.
MacBook


Developers
Platform
Android Studio
Google Play
Jetpack
Search
Documentation
OVERVIEW
GUIDES
REFERENCE
SAMPLES
DESIGN & QUALITY
Test Support Library
Wearable Library
Play Billing Library
Play Install Referrer Library
Android Things
Kotlin API Reference (Preview)
Android Platform
AndroidX
API reference
You can build your Android app with the Android Platform APIs and the following libraries.
Developing with Kotlin? Check out the Kotlin reference for the Android Platform and AndroidX library.
Android Support Library
Provides a variety of Android feature and utility APIs that are compatible with a wide range of platform versions.
AndroidX
Refactored versions of the Android APIs that are not bundled with the operating system.
MacBook


Developers
Platform
Android Studio
Google Play
Jetpack
Search
Documentation
OVERVIEW
GUIDES
REFERENCE
SAMPLES
DESIGN & QUALITY
Test Support Library
Wearable Library
Play Billing Library
Play Install Referrer Library
Android Things
Kotlin API Reference (Preview)
Android Platform
AndroidX
API reference
You can build your Android app with the Android Platform APIs and the following libraries.
Developing with Kotlin? Check out the Kotlin reference for the Android Platform and AndroidX library.
Android Support Library
Provides a variety of Android feature and utility APIs that are compatible with a wide range of platform versions.
AndroidX
Refactored versions of the Android APIs that are not bundled with the operating system.
MacBook


Developers
Platform
Android Studio
Google Play
Jetpack
Search
OVERVIEW
GUIDES
REFERENCE
SAMPLES
DESIGN & QUALITY
androidx.fragment.app
Overview
Classes
Interfaces
Exceptions
androidx.gridlayout.widget
androidx.heifwriter
androidx.interpolator.view.animation
androidx.leanback.app
androidx.leanback.database
androidx.leanback.graphics
androidx.leanback.media
androidx.leanback.preference
androidx.leanback.system
androidx.leanback.widget
androidx.leanback.widget.picker
androidx.lifecycle
androidx.loader.app
FragmentHostCallback
Integration points with the Fragment host.
Fragment
Static library support version of the framework's android.app.Fragment.
DialogFragment
Static library support version of the framework's android.app.DialogFragment.
FragmentManagerNonConfig
FragmentManagerNonConfig stores the retained instance fragments across activity recreation events.
Extension functions summary
For FragmentManager
Unit FragmentManager.transaction(now: Boolean = false, allowStateLoss: Boolean = false, body: FragmentTransaction.() -> Unit)
Run body in a FragmentTransaction which is automatically committed if it completes without exception.
MacBook

Building Kotlin-friendly libraries

Building Kotlin-friendly libraries

- Port public API or entire library to Kotlin

Building Kotlin-friendly libraries

- Port public API or entire library to Kotlin
- Ship sibling artifact with Kotlin extensions

Building Kotlin-friendly libraries

- Port public API or entire library to Kotlin
- Ship sibling artifact with Kotlin extensions
- ???

```kotlin
inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.isDigitsOnly()
```

```
class TextUtils {
    static boolean isDigitsOnly(CharSequence str) {
        int len = str.length();
        // …
    }
}

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.isDigitsOnly()
```

```
class TextUtils {
    static boolean isDigitsOnly(@NonNull CharSequence str) {
        int len = str.length();
        // …
    }
}

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.isDigitsOnly()
```

```
class TextUtils {
    static boolean isDigitsOnly(@NonNull CharSequence str) {
        int len = str.length();
        // …
    }
}

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.isDigitsOnly()
```

```
class TextUtils {
    @ExtensionFunction
    static boolean isDigitsOnly(@NonNull CharSequence str) {
        int len = str.length();
        // …
    }
}

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.isDigitsOnly()
```

```
class TextUtils {
    @ExtensionFunction
    static boolean isDigitsOnly(@NonNull CharSequence str) {
        int len = str.length();
        // …
    }
}

inline fun CharSequence.isDigitsOnly() = TextUtils.isDigitsOnly(this)

val onlyDigits = phoneNumber.isDigitsOnly()
```

```
class TextUtils {
    @ExtensionFunction
    static boolean isDigitsOnly(@NonNull CharSequence str) {
        int len = str.length();
        // …
    }
}

val onlyDigits = phoneNumber.isDigitsOnly()
```

```
class TextUtils {
    @ExtensionFunction
    static boolean isDigitsOnly(@NonNull CharSequence str) {
        int len = str.length();
        // …
    }
}

val onlyDigits = phoneNumber.isDigitsOnly()
```

```
class TextUtils {
    @ExtensionFunction
    static boolean isDigitsOnly(@NonNull CharSequence str) {
        int len = str.length();
        // …
    }
}

val onlyDigits = phoneNumber.isDigitsOnly()
// in the bytecode we get
val onlyDigits = TextUtils.isDigitsOnly(phoneNumber)
```

```kotlin
inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.updatePadding(left = 10, right = 10)
```

```kotlin
inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.updatePadding(left = 10, right = 10)
```

```
class View {
    void setPadding(int left, int top, int right, int bottom) { /* … */ }
}

inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}

avatarView.updatePadding(left = 10, right = 10)
```

```
class View {
    void setPadding(
        @KtName("left") int left,
        @KtName("top") int top,
        @KtName("right") int right,
        @KtName("bottom") int bottom
    ) { /* … */ }
}

inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}
```

```
class View {
    void setPadding(
        @KtName("left") @DefaultValue("paddingLeft") int left,
        @KtName("top") @DefaultValue("paddingTop") int top,
        @KtName("right") @DefaultValue("paddingRight") int right,
        @KtName("bottom") @DefaultValue("paddingBottom") int bottom
    ) { /* … */ }
}

inline fun View.updatePadding(
    left: Int = paddingLeft,
    top: Int = paddingTop,
    right: Int = paddingRight,
    bottom: Int = paddingBottom
) {
    setPadding(left, top, right, bottom)
}
```

```
class View {
    void setPadding(
        @KtName("left") @DefaultValue("paddingLeft") int left,
        @KtName("top") @DefaultValue("paddingTop") int top,
        @KtName("right") @DefaultValue("paddingRight") int right,
        @KtName("bottom") @DefaultValue("paddingBottom") int bottom
    ) { /* … */ }
}

avatarView.updatePadding(left = 10, right = 10)
```

```
class View {
    void setPadding(
        @KtName("left") @DefaultValue("paddingLeft") int left,
        @KtName("top") @DefaultValue("paddingTop") int top,
        @KtName("right") @DefaultValue("paddingRight") int right,
        @KtName("bottom") @DefaultValue("paddingBottom") int bottom
    ) { /* … */ }
}

avatarView.setPadding(left = 10, right = 10)
```

```
class View {
    void setPadding(
        @KtName("left") @DefaultValue("paddingLeft") int left,
        @KtName("top") @DefaultValue("paddingTop") int top,
        @KtName("right") @DefaultValue("paddingRight") int right,
        @KtName("bottom") @DefaultValue("paddingBottom") int bottom
    ) { /* … */ }
}

avatarView.setPadding(left = 10, right = 10)
// in bytecode we get
avatarView.setPadding(
    10, avatarView.paddingTop, 10, avatarView.paddingBottom)
```

KEEP-110

KEEP-110

- `@ExtensionFunction` / `@ExtensionProperty` — Turn a static method with at least one argument into an extension function or an extension property.

KEEP-110

- `@ExtensionFunction` / `@ExtensionProperty` — Turn a static method with at least one argument into an extension function or an extension property.
- `@DefaultValue` — Default parameter values.

KEEP-110

- `@ExtensionFunction` / `@ExtensionProperty` — Turn a static method with at least one argument into an extension function or an extension property.
- `@DefaultValue` — Default parameter values.
- `@KtName` — An alternate name for methods, fields, and parameters for use by Kotlin code.

Building Kotlin-friendly libraries

- Port public API or entire library to Kotlin
- Ship sibling artifact with Kotlin extensions
- ???

Building Kotlin-friendly libraries

- Port public API or entire library to Kotlin
- Ship sibling artifact with Kotlin extensions
- KEEP-110 annotations

Android KTX

core-ktx → core → Android framework

fragment-ktx → fragment

palette-ktx → palette

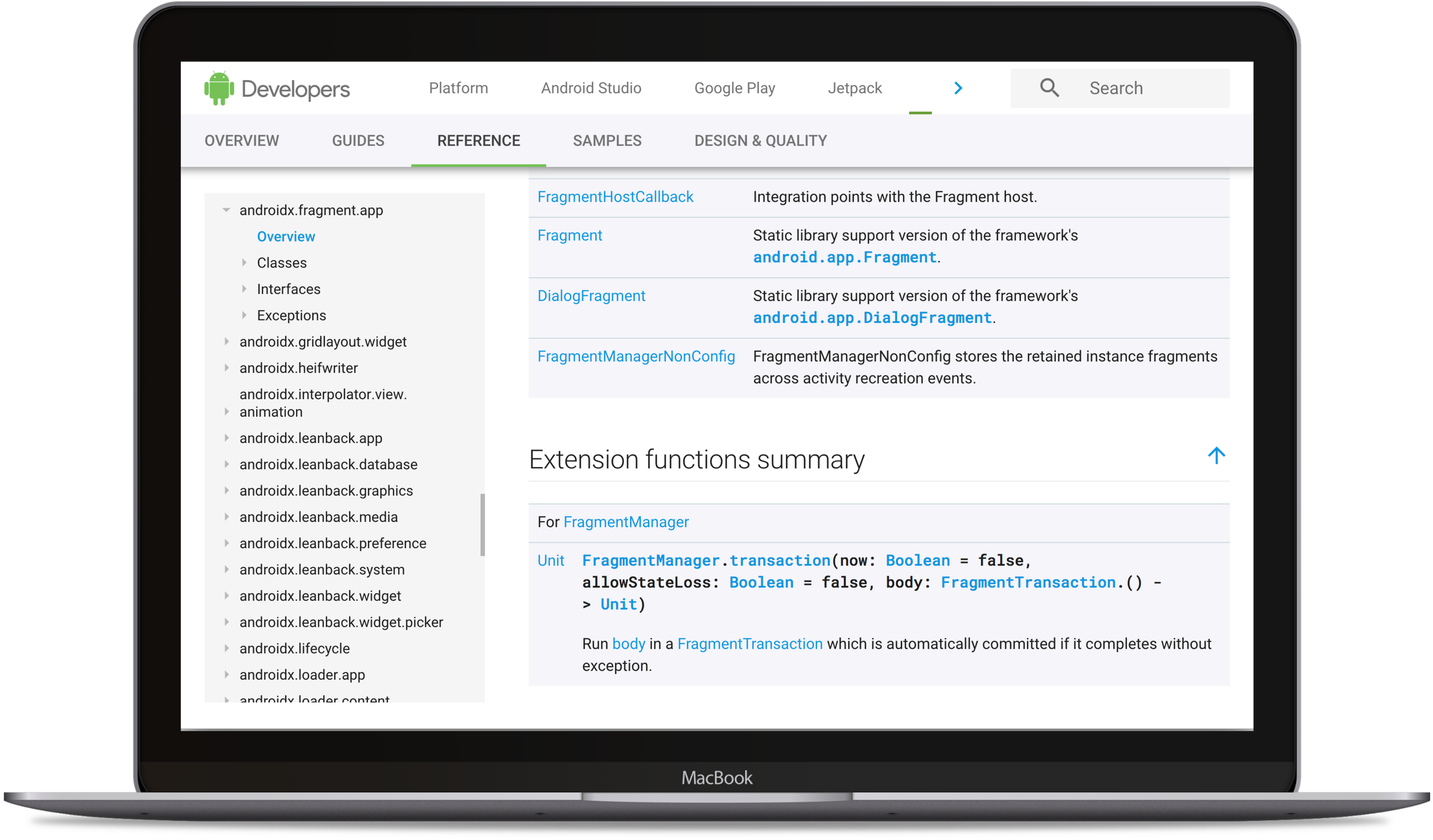
collection-ktx → collection

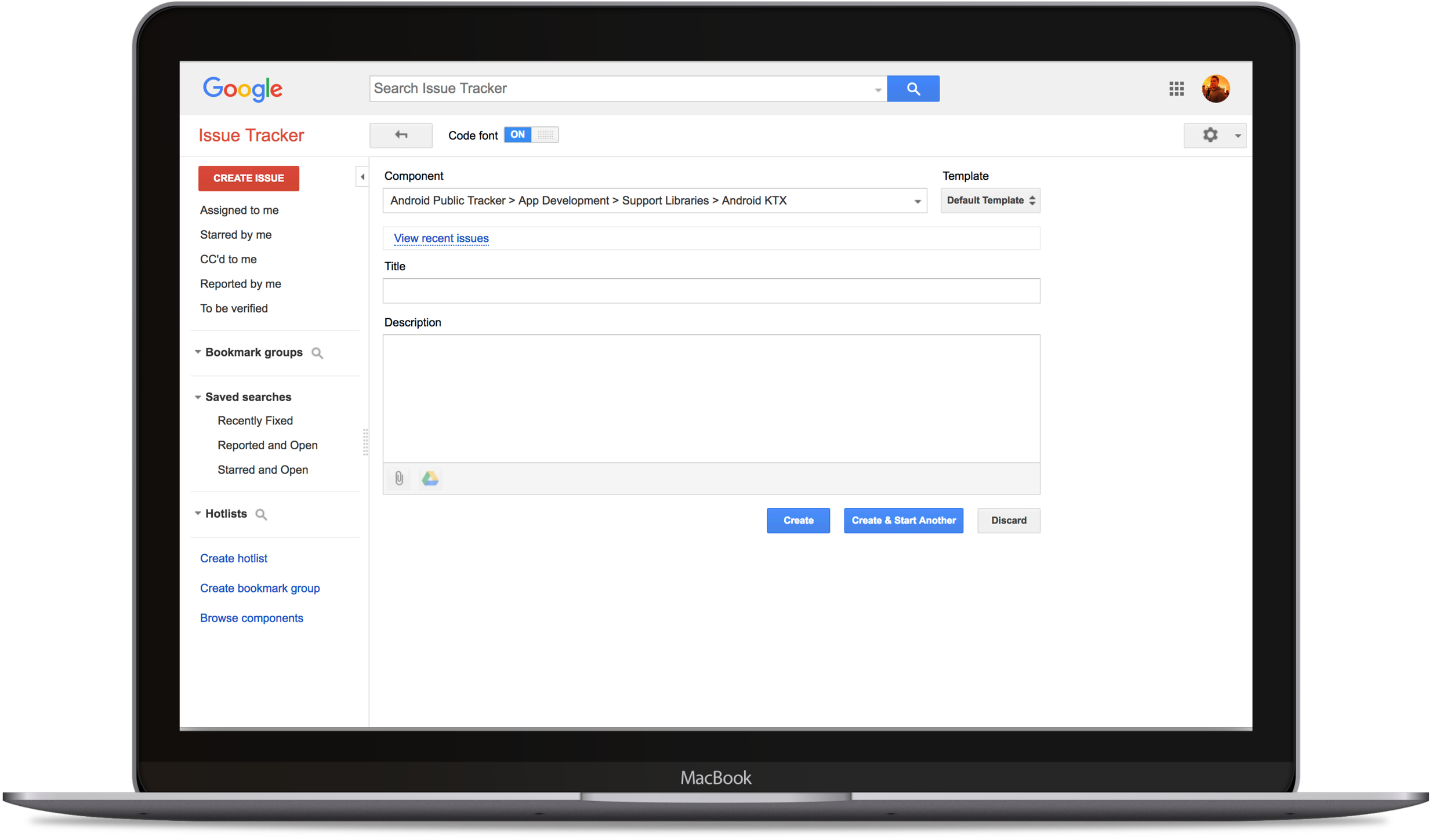
lifecycle-reactivestreams-ktx → lifecycle-reactivestreams

sqlite-ktx → sqlite

navigation-*-ktx → navigation-*

work-runtime-ktx → work-runtime


Developers
Platform
Android Studio
Google Play
Jetpack
Search
OVERVIEW
GUIDES
REFERENCE
SAMPLES
DESIGN & QUALITY
androidx.fragment.app
Overview
Classes
Interfaces
Exceptions
androidx.gridlayout.widget
androidx.heifwriter
androidx.interpolator.view.animation
androidx.leanback.app
androidx.leanback.database
androidx.leanback.graphics
androidx.leanback.media
androidx.leanback.preference
androidx.leanback.system
androidx.leanback.widget
androidx.leanback.widget.picker
androidx.lifecycle
androidx.loader.app
FragmentHostCallback
Integration points with the Fragment host.
Fragment
Static library support version of the framework's android.app.Fragment.
DialogFragment
Static library support version of the framework's android.app.DialogFragment.
FragmentManagerNonConfig
FragmentManagerNonConfig stores the retained instance fragments across activity recreation events.
Extension functions summary
For FragmentManager
Unit FragmentManager.transaction(now: Boolean = false, allowStateLoss: Boolean = false, body: FragmentTransaction.() -> Unit)
Run body in a FragmentTransaction which is automatically committed if it completes without exception.
MacBook


Google
Search Issue Tracker
Issue Tracker
Code font
ON
CREATE ISSUE
Assigned to me
Starred by me
CC'd to me
Reported by me
To be verified
Bookmark groups
Saved searches
Recently Fixed
Reported and Open
Starred and Open
Hotlists
Create hotlist
Create bookmark group
Browse components
Component
Android Public Tracker > App Development > Support Libraries > Android KTX
Template
Default Template
View recent issues
Title
Description
Create
Create & Start Another
Discard
MacBook

Component

Android Public Tracker > App Development > Support Libraries > Android KTX

Template

Default Template

View recent issues

Title

Description

Create | Create & Start Another | Discard

Java annotations for Kotlin sugar

- **Type:** Design proposal
- **Author:** Jake Wharton
- **Status:** Under consideration
- **Prototype:** Partially implemented in Kotlin 1.1.60

Feedback

Discussion of this proposal is held in this issue.

Summary

This proposal adds annotations which enable the use of more Kotlin language features like extension functions, extension properties, default values, and property names for Java code.

- `@ExtensionFunction` / `@ExtensionProperty` - Turn a static method with at least one argument into an extension function or an extension property.
- `@DefaultValue` - Default parameter values.
- `@KtName` - An alternate name for methods, fields, and parameters for use by Kotlin code.

Motivation / use cases

There is a large corpus of Java libraries which, for various reasons, can't or won't port to Kotlin for the foreseeable future. This fact does not render them as obsolete or inferior to a Kotlin library. In fact, a certain portion of Java library authors are sympathetic to Kotlin users and are willing to do more for compatibility.

https://github.com/Kotlin/KEEP/issues/110

Java annotations for Kotlin sugar

- **Type**: Design proposal
- **Author**: Jake Wharton
- **Status**: Under consideration
- **Prototype**: Partially implemented in Kotlin 1.1.60

Feedback

Discussion of this proposal is held in this issue.

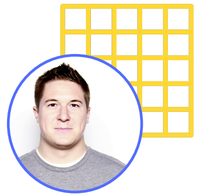
Summary

This proposal adds annotations which enable the use of more Kotlin language features like extension functions, extension properties, default values, and property names for Java code.

- `@ExtensionFunction` / `@ExtensionProperty` - Turn a static method with at least one argument into an extension function or an extension property.
- `@DefaultValue` - Default parameter values.
- `@KtName` - An alternate name for methods, fields, and parameters for use by Kotlin code.

Motivation / use cases

Thank you



Jake Wharton

@JakeWharton