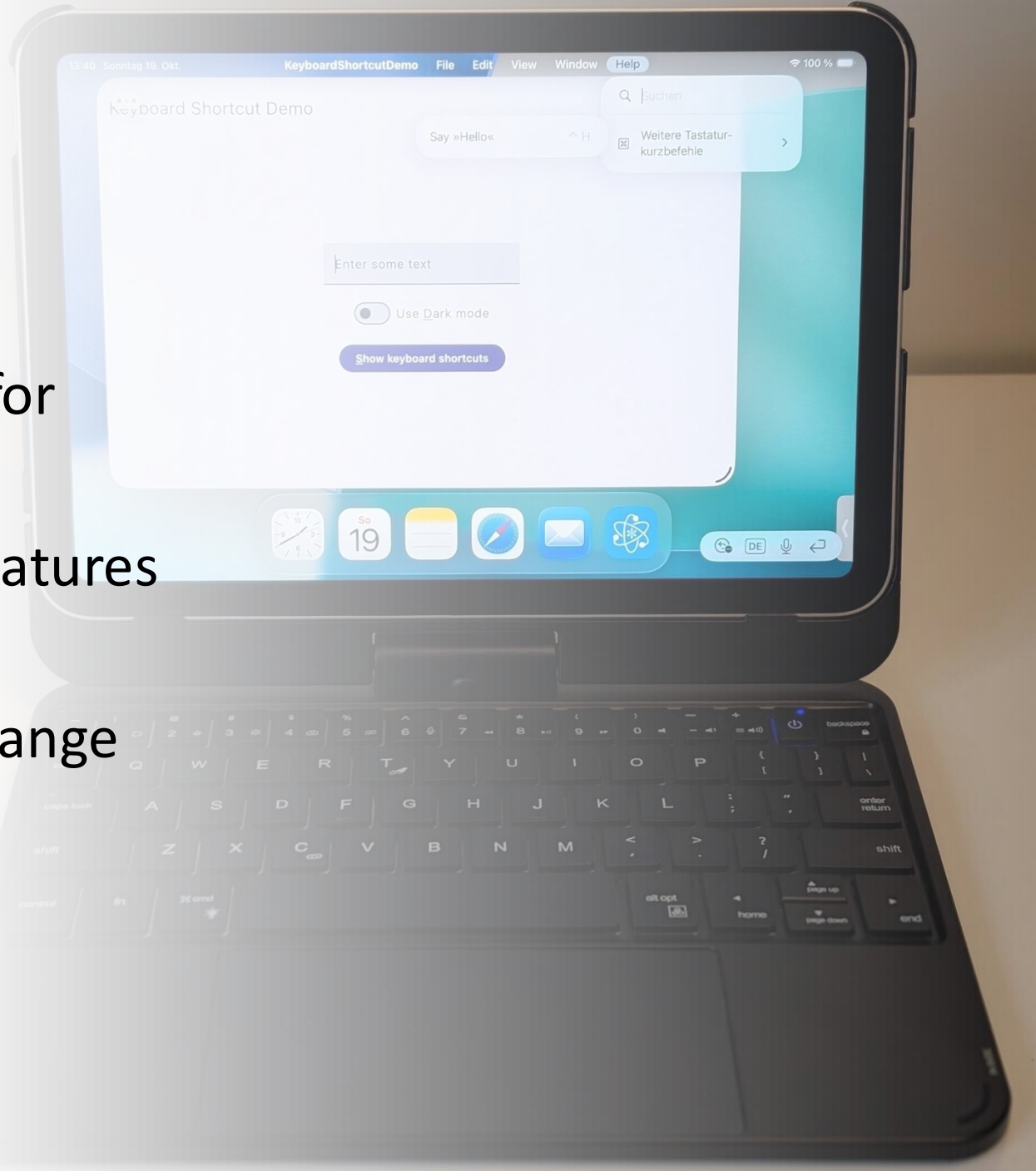
Key to success - understanding keyboard shortcuts in Jetpack Compose

Thomas Künneth

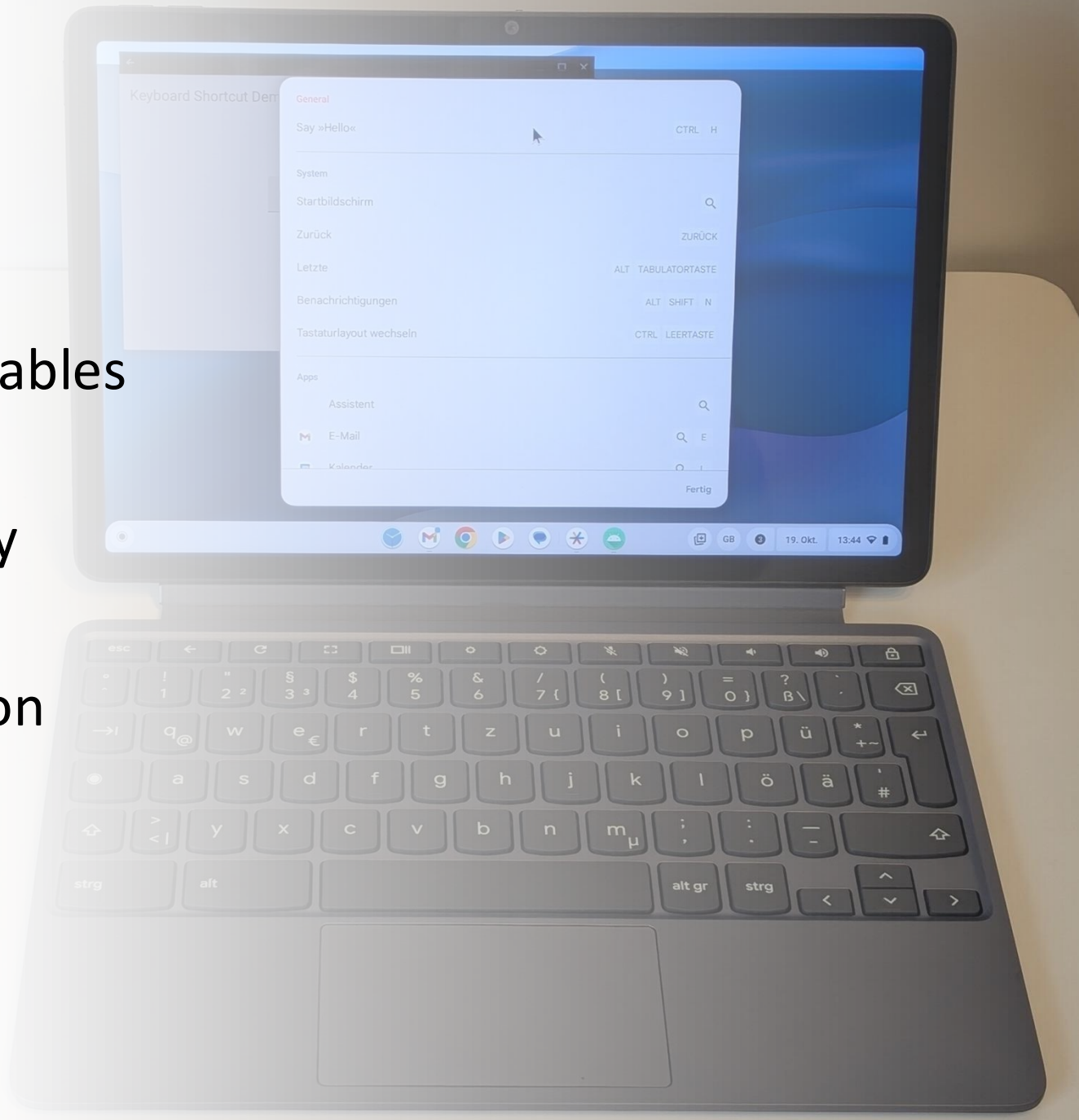
Experts

- Apples released several keyboards for various iPad models
- iPadOS gained many productivity features that utilize the keyboards
- Independent vendors offer a wide range of iPad keyboard solutions



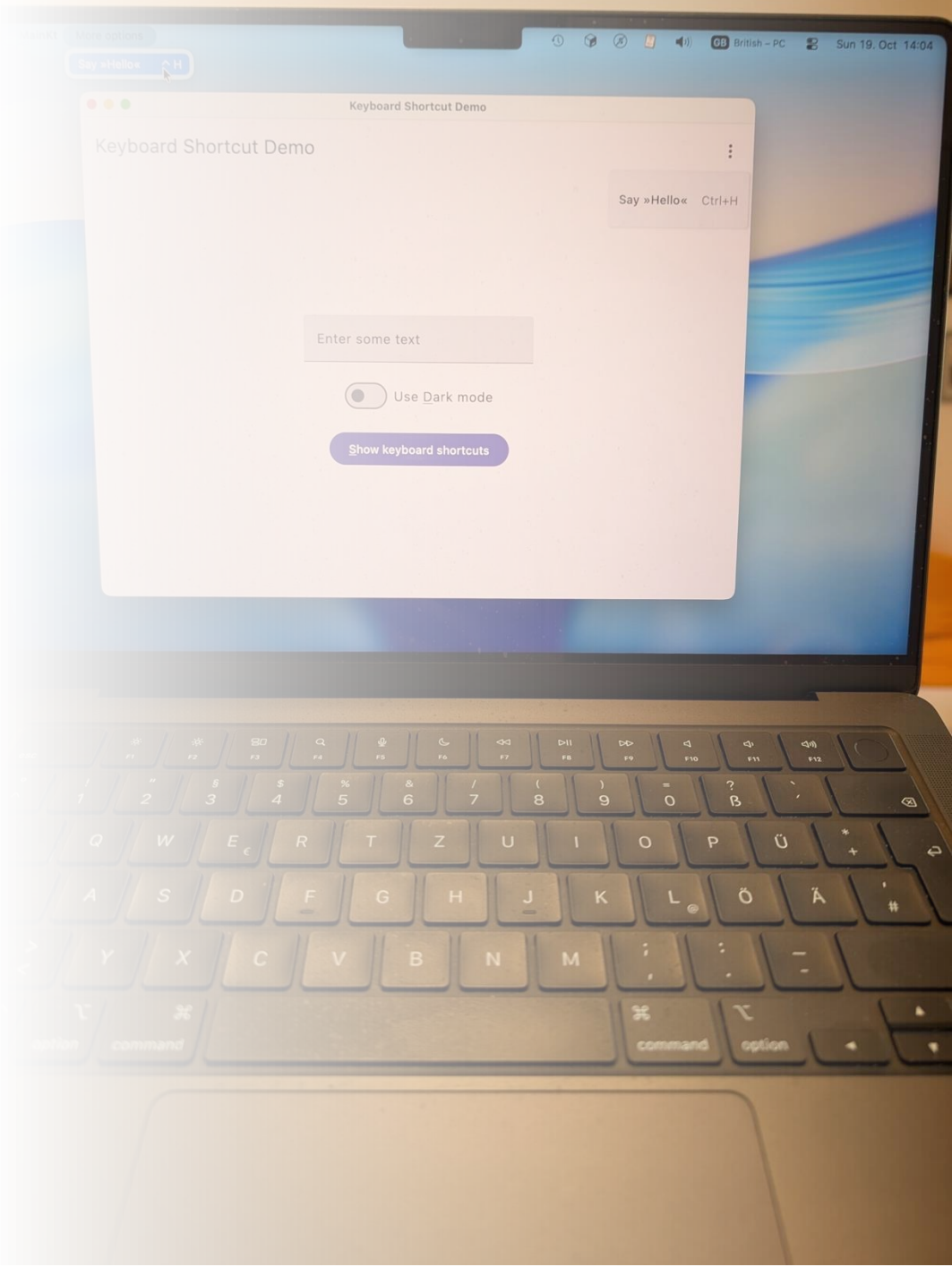
Experts

- ChromeOS has embraced detachables since 2018 (HP Chromebook x2)
- The 2020 Lenovo Duet was a very popular budget 2 in 1 device
- ChromeOS UI evolved to transition seamlessly between tablet- and desktop mode

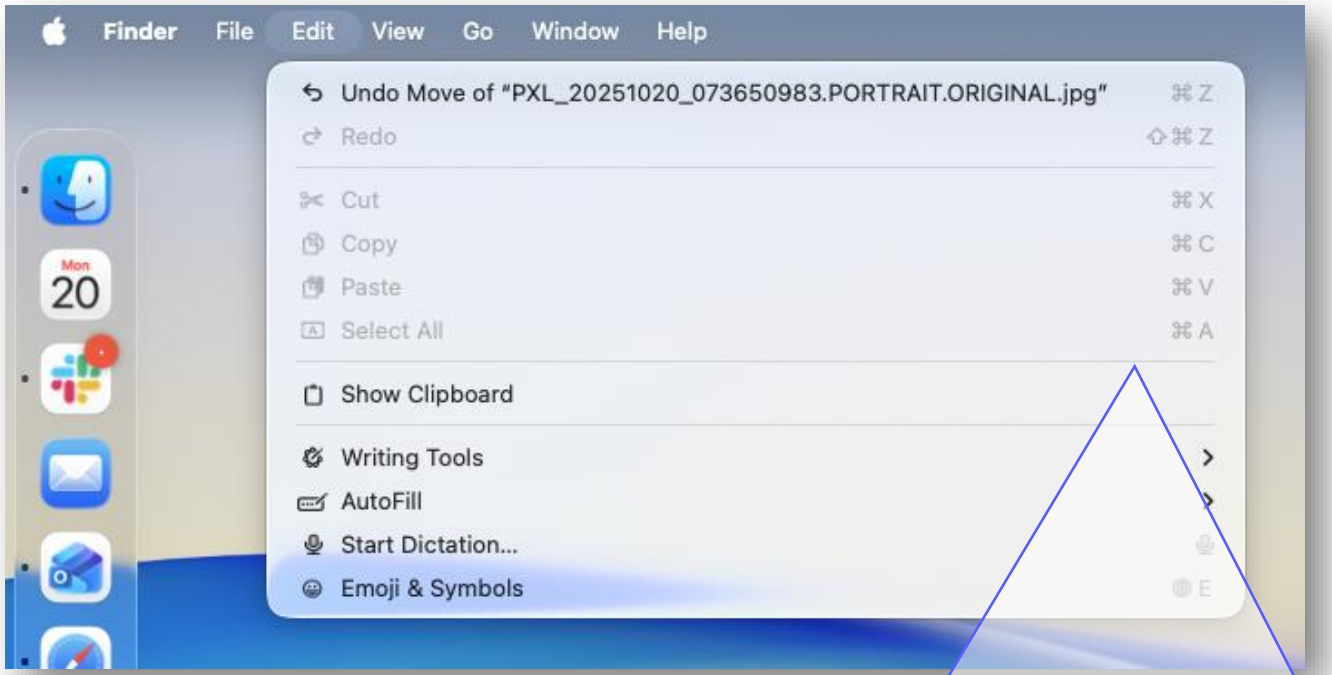


Experts

- Desktop operating systems have been relying on physical keyboards from the beginning
- With the rise of graphical user interfaces, they combined the **ease of use of the mouse** with the **speed of using a keyboard**



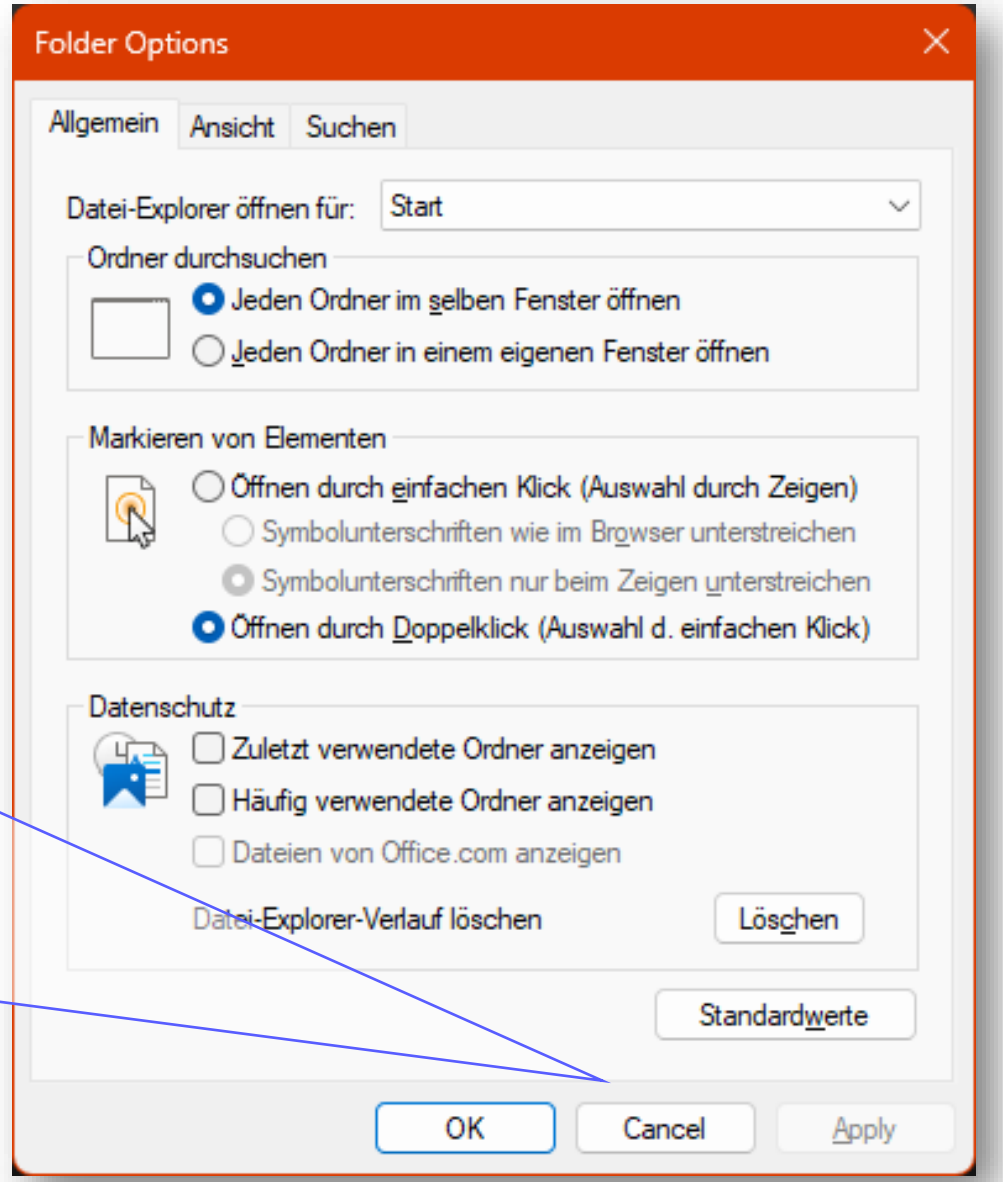
Experts



- **Shortcuts** execute commands
- Must be discoverable (e.g. menus, tooltips)
- Alternative way of invoking a command (usually shouldn’t be the only method)

SnappMobile

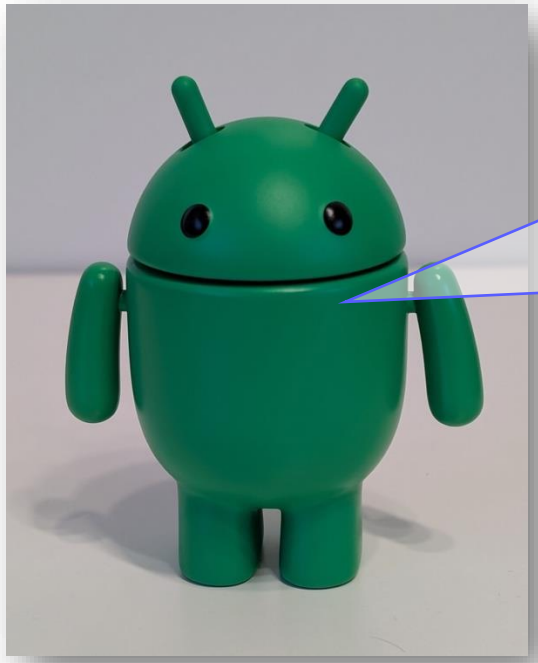
- **Access keys** activate visible UI elements
- Often, combination of *Alternate* (*Option* on macOS) and a letter (e.g. Alt+A)
- Should be discoverable on the element itself (e.g. the underlined letter in *Apply*)
- Alternative way of clicking a control



SnappMobile

- **Hotkeys** trigger global or system-wide actions
- Shortcuts that work across applications (not tied to the focused app's UI)
- Usually **not** discoverable in menus (e.g. Command+Tab)

SnappMobile

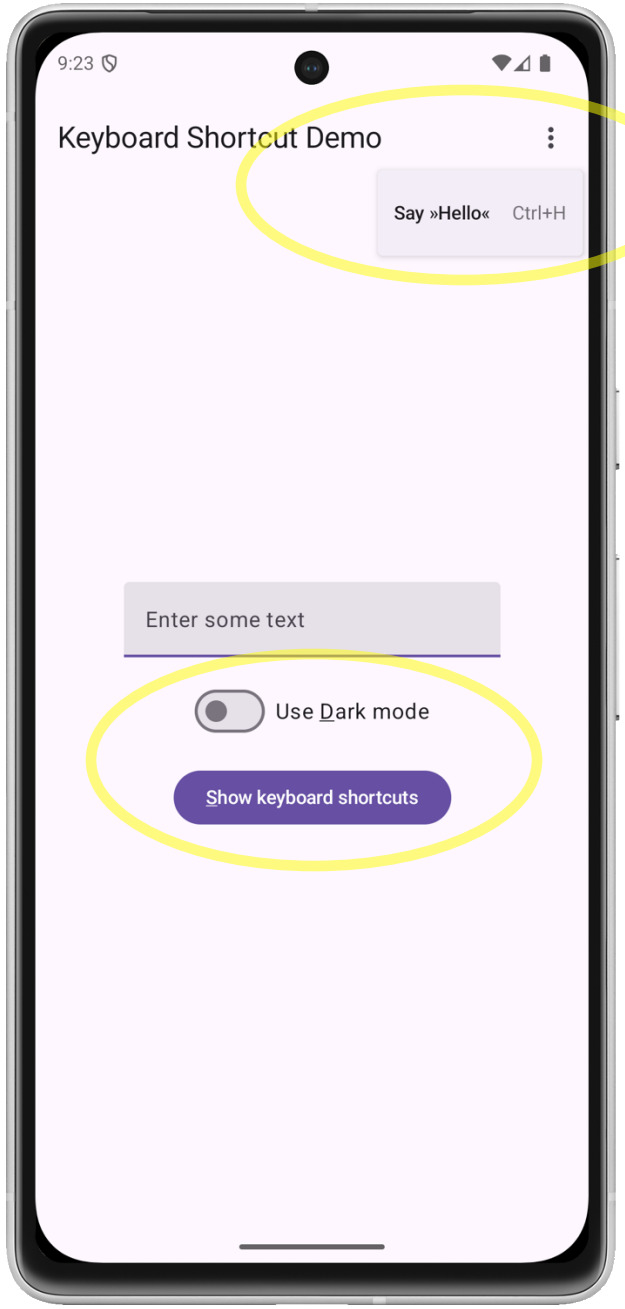


Sounds cool

But how are these types of key combinations reflected in Jetpack Compose apps?

Shortcuts

Access keys



https://github.com/tkuenneth/KeyboardShortcutDemo

Experts

SnappMobile

No traditional menu bar on Android

- App bar (action bar) contains top-level commands (action items)
- Much like menu items, they can have a label, an icon, and a tooltip
- Can be put in a menu
- Compose app bar implementation has no built-in support for keyboard shortcuts
- `View`-based action bar implementation has (invoked from `Activity` code)

`Activity`-level shortcut support

- `onProvideKeyboardShortcuts()` informs the system about **keyboard shortcut groups**; shortcuts that belong to the same category are shown below a headline
- `onKeyShortcut()` is called by the system when a registered keyboard shortcut has been invoked

```java
@Override
public void onProvideKeyboardShortcuts(
        List<KeyboardShortcutGroup> data, Menu menu, int deviceId) {
    if (menu == null) {
      return;
    }
    KeyboardShortcutGroup group = null;
    int menuSize = menu.size();
    for (int i = 0; i < menuSize; ++i) {
        final MenuItem item = menu.getItem(i);
        final CharSequence title = item.getTitle();
        final char alphaShortcut = item.getAlphabeticShortcut();
        final int alphaModifiers = item.getAlphabeticModifiers();
        if (title != null && alphaShortcut != MIN_VALUE) {
            if (group == null) {
                final int resource = mApplication.getApplicationInfo().labelRes;
                group = new KeyboardShortcutGroup(resource != 0 ? getString(resource) : null);
            }
            group.addItem(new KeyboardShortcutInfo(
                title, alphaShortcut, alphaModifiers));
        }
    }
    if (group != null) {
        data.add(group);
    }
}
```

If there's a menu, and if the menu items have shortcuts, add them

If there's an action bar, let it handle the keyboard shortcuts

> Called when a key shortcut event is not handled by any of the views in the Activity. Override this method to implement global key shortcuts for the Activity. Key shortcuts can also be implemented by setting the `shortcut` property of menu items.
>
> Params: `keyCode` – The value in event.getKeyCode().
> `event` – Description of the key event.
>
> Returns: True if the key shortcut was handled.

```java
public boolean onKeyShortcut(int keyCode, KeyEvent event) {
    // Let the Action Bar have a chance at handling the shortcut.
    ActionBar actionBar = getActionBar();
    return (actionBar != null && actionBar.onKeyShortcut(keyCode, event));
}
```

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    listKeyboardShortcutInfo = globalShortcuts.map { shortcut ->
        KeyboardShortcutInfo(
            shortcut.label,
            shortcut.keyAsString.first(),
            shortcut.modifiers()
        )
    }
    updateFlows()
    setContent {
        val hardKeyboardHidden by hardKeyboardHiddenFlow.collectAsStateWithLifecycle()
        val systemInDarkMode = isSystemInDarkTheme()
        var darkMode by rememberSaveable { mutableStateOf(systemInDarkMode) }
        MaterialTheme(colorScheme = if (darkMode) darkColorScheme() else lightColorScheme()) {
            MainScreen(
                listKeyboardShortcuts = globalShortcuts,
                hardKeyboardHidden = hardKeyboardHidden == Configuration.HARDKEYBOARDHIDDEN_YES,
                darkMode = darkMode,
                showKeyboardShortcuts = { requestShowKeyboardShortcuts() },
                toggleDarkMode = { darkMode = !darkMode }
            )
        }
    }
}
```

```
package android.view;
```

```kotlin
listOf(
    KeyboardShortcut(
        label = getString(Res.string.say_hello),
        key = Key.H,
        keyAsString = "H",
        ctrl = true
    )
)
```

```
package dev.tkuenneth.keyboardshortcutdemo
```

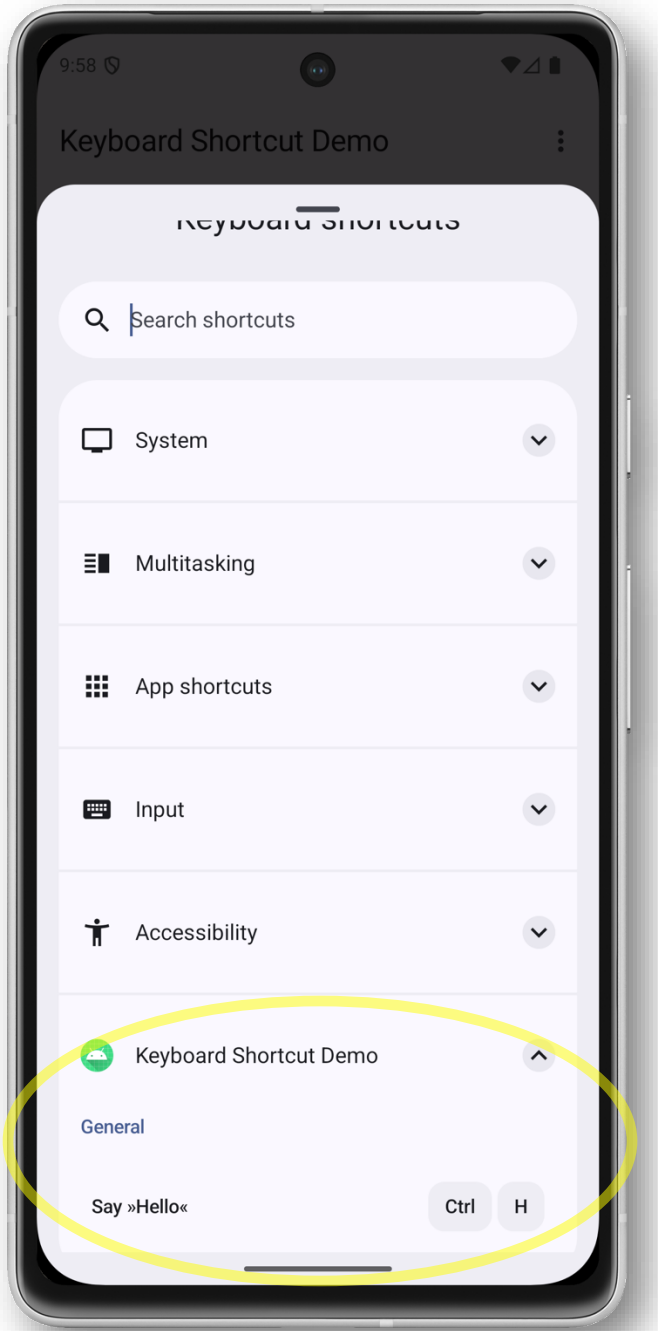
Can’t easily get it from `Key`

SnappMobile

Tell Android about our shortcuts

```kotlin
override fun onProvideKeyboardShortcuts(
    data: MutableList<KeyboardShortcutGroup>,
    menu: Menu?,
    deviceId: Int
) {
    super.onProvideKeyboardShortcuts(data, menu, deviceId)
    data.add(KeyboardShortcutGroup(
        getString(R.string.general),
        listKeyboardShortcutInfo
    ))
}
```

Keyboard shortcuts are grouped, based on their category



Experts

```
Request the Keyboard Shortcuts screen to show up. This will trigger
onProvideKeyboardShortcuts to retrieve the shortcuts for the foreground activity.

public final void requestShowKeyboardShortcuts() {
    final ComponentName sysuiComponent = ComponentName.unflattenFromString(
            getResources().getString(
                    com.android.internal.R.string.config_systemUIServiceComponent));
    Intent intent = new Intent(Intent.ACTION_SHOW_KEYBOARD_SHORTCUTS);
    intent.setPackage(sysuiComponent.getPackageName());
    sendBroadcastAsUser(intent, Process.myUserHandle());
}
```

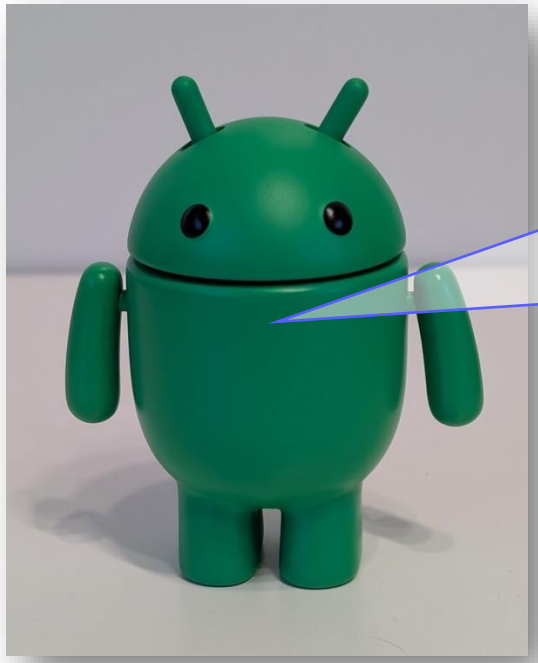
```
1 MainScreen(
2     listKeyboardShortcuts = globalShortcuts,
3     hardKeyboardHidden = hardKeyboardHidden == Configuration.HARDKEYBOARDHIDDEN_YES,
4     darkMode = darkMode,
5     showKeyboardShortcuts = ::requestShowKeyboardShortcuts,
6     toggleDarkMode = { darkMode = !darkMode }
7 )
```

Get notified when one of our shortcuts is typed

```kotlin
override fun onKeyShortcut(
    keyCode: Int, event: KeyEvent
): Boolean {
    val displayLabel = event.displayLabel.uppercase()
    globalShortcuts.forEach { shortcut ->
        if (shortcut.keyAsString == displayLabel &&
            event.hasModifiers(shortcut.modifiers())
        ) {
            shortcut.triggerAction()
            return true
        }
    }
    return super.onKeyShortcut(keyCode, event)
}
```

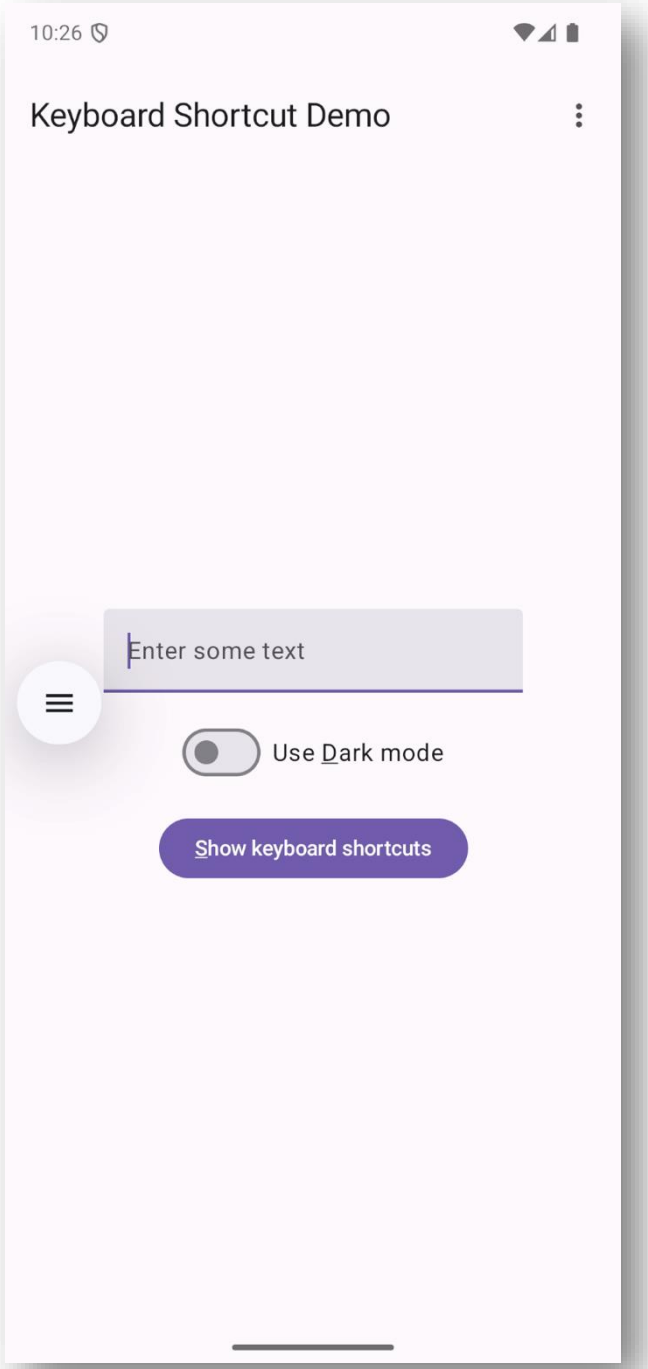
```kotlin
private fun KeyboardShortcut.modifiers(): Int {
    var mask = 0
    if (ctrl) mask = mask or KeyEvent.META_CTRL_ON
    if (meta) mask = mask or KeyEvent.META_META_ON
    if (alt) mask = mask or KeyEvent.META_ALT_ON
    if (shift) mask = mask or KeyEvent.META_SHIFT_ON
    return mask
}
```

```kotlin
data class KeyboardShortcut(
    val label: String,
    val key: Key,
    val keyAsString: String,
    val ctrl: Boolean = false,
    val meta: Boolean = false,
    val alt: Boolean = false,
    val shift: Boolean = false,
) {
    private val channel = Channel<Unit>(Channel.CONFLATED)
    val flow = channel.receiveAsFlow()

    fun triggerAction() {
        channel.trySend(Unit)
    }

    val shortcutAsText: String
        get() {
            val parts = mutableListOf<String>()
            if (ctrl) {
                parts.add("Ctrl")
            }
            if (meta) {
                parts.add("Meta")
            }
            if (alt) {
                parts.add("Alt")
            }
            if (shift) {
                parts.add("Shift")
            }
            parts.add(keyAsString)
            return parts.joinToString("+")
        }
}
```



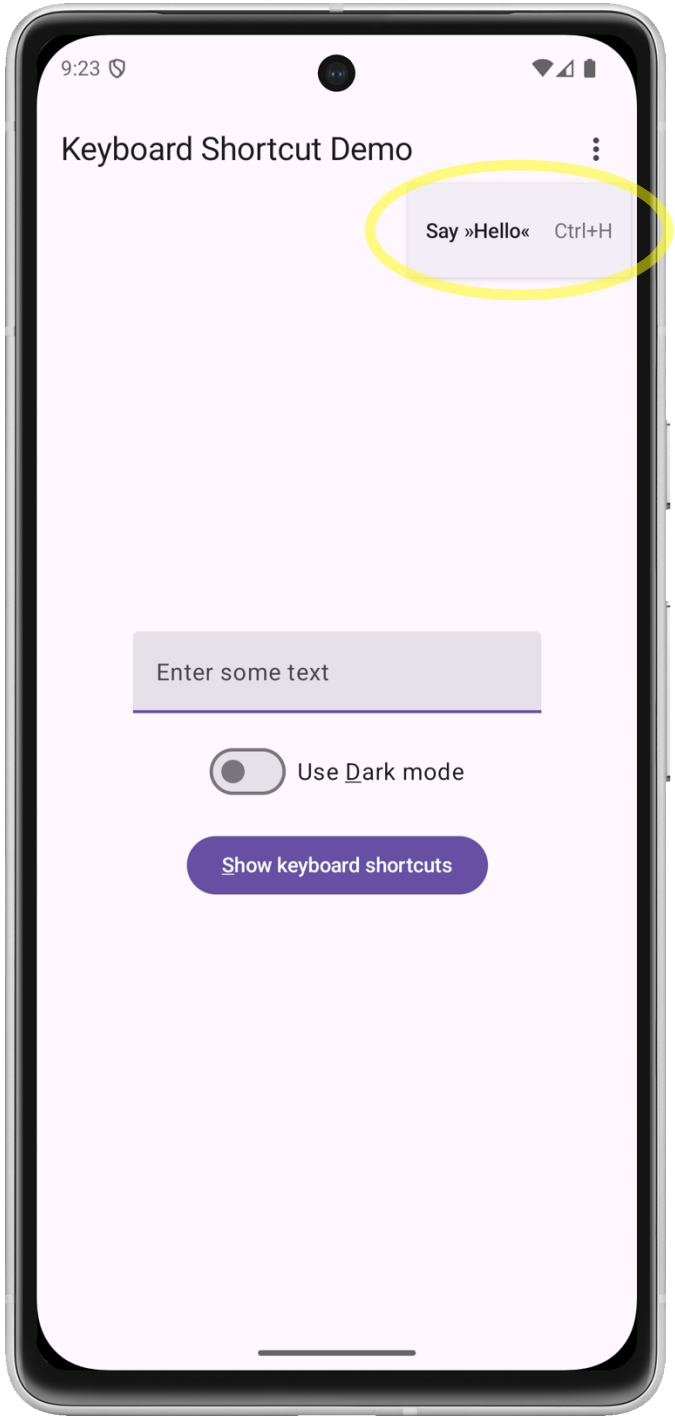
Looks cool

But how do we handle the shortcuts?




```kotlin
@Composable
fun MainScreen(
    listKeyboardShortcuts: List<KeyboardShortcut>,
    hardKeyboardHidden: Boolean,
    darkMode: Boolean,
    showKeyboardShortcuts: () -> Unit,
    toggleDarkMode: () -> Unit,
) {
    var snackbarMessage by remember { mutableStateOf("") }
    val helloMessage = stringResource(Res.string.hello)
    LaunchedEffect(listKeyboardShortcuts) {
        listKeyboardShortcuts.forEach { shortcut ->
            launch {
                shortcut.flow.collectLatest {
                    snackbarMessage = helloMessage
                }
            }
        }
    }
    KeyboardShortcutDemo(
        hardwareKeyboardHidden = hardKeyboardHidden,
        snackbarMessage = snackbarMessage,
        shortcuts = listKeyboardShortcuts,
        darkMode = darkMode,
        showKeyboardShortcuts = showKeyboardShortcuts,
        clearSnackbarMessage = { snackbarMessage = "" },
        toggleDarkMode = toggleDarkMode,
    )
}
```

Experts

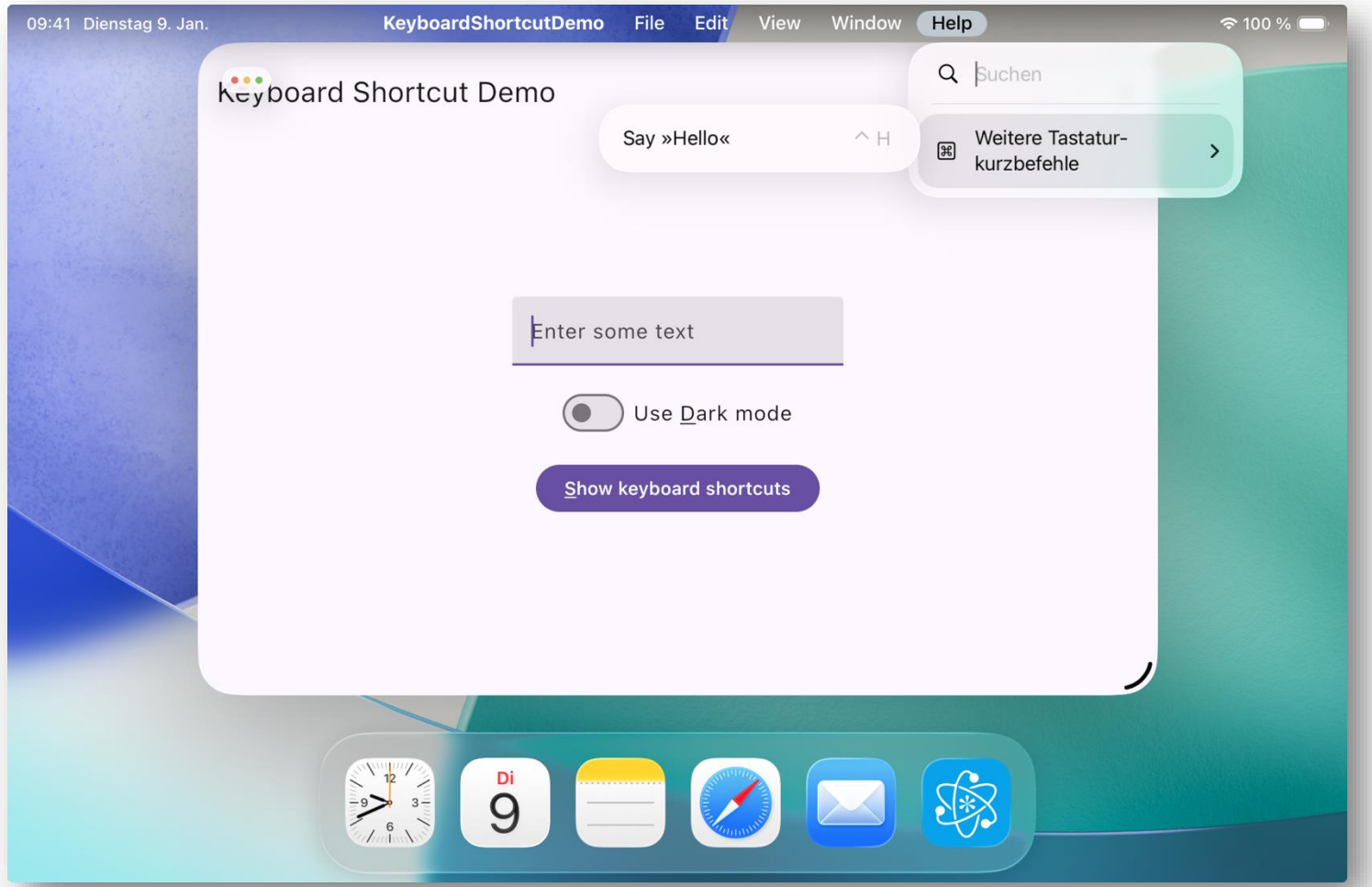



```
TopAppBar(
    title = { Text(stringResource(Res.string.app_name)) },
    actions = {
        IconButton(onClick = { showMenu = !showMenu }) {
            Icon(
                imageVector = Icons.Filled.MoreVert,
                contentDescription = stringResource(Res.string.more_options)
            )
        }
        DropdownMenu(
            expanded = showMenu,
            onDismissRequest = { showMenu = false }
        ) {
            shortcuts.forEach { shortcut ->
                DropdownMenuItemWithShortcut(
                    text = shortcut.label,
                    shortcut = shortcut.shortcutAsText,
                    onClick = {
                        showMenu = false
                        shortcut.triggerAction()
                    }
                )
            }
        }
    }
)
```

```
val shortcutAsText: String 3 Usages
    get() {
        val parts = mutableListOf<String>()
        if (ctrl) {
            parts.add("Ctrl")
        }
        if (meta) {
            parts.add("Meta")
        }
        if (alt) {
            parts.add("Alt")
        }
        if (shift) {
            parts.add("Shift")
        }
        parts.add(keyAsString)
        return parts.joinToString( separator = "+")
    }
```

```kotlin
@Composable
fun DropdownMenuItemWithShortcut(
    text: String,
    shortcut: String?,
    onClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    DropdownMenuItem(
        text = {
            ShortcutText(
                text = text,
                shortcut = shortcut
            )
        },
        onClick = onClick,
        modifier = modifier
    )
}
```

```kotlin
@Composable
fun ShortcutText(
    text: String,
    shortcut: String?,
) {
    Row(
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.SpaceBetween
    ) {
        Text(
            text = text,
            modifier = Modifier.alignByBaseline()
        )
        shortcut?.let { shortcut ->
            Text(
                text = shortcut,
                style = MaterialTheme.typography.bodyMedium,
                color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.6f),
                modifier = Modifier
                    .alignByBaseline()
                    .padding(start = 16.dp)
            )
        }
    }
}
```

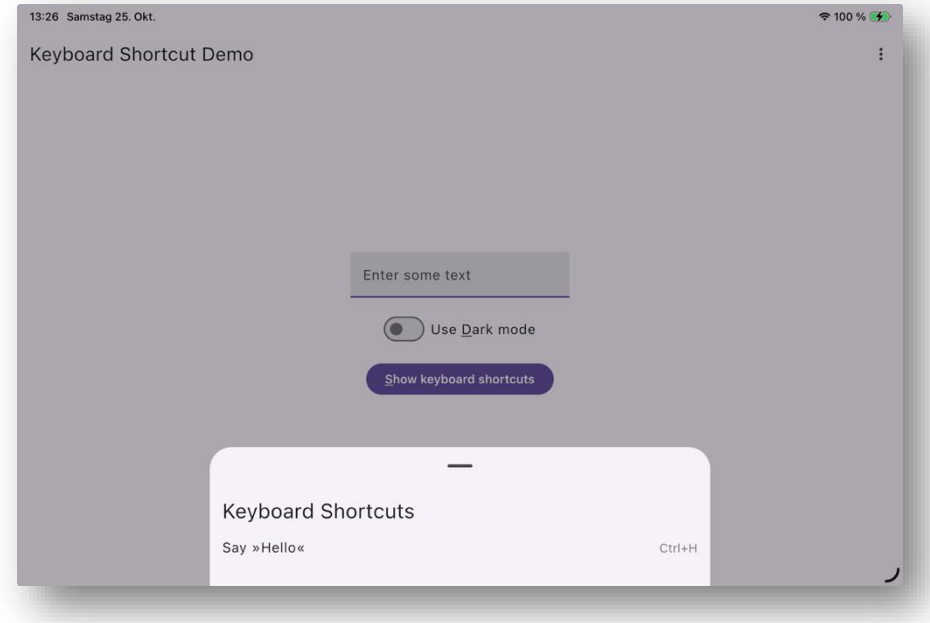


Experts

SnappMobile

```
fun MainViewController(): UIViewController {
    return KeyboardShortcutViewController(
        shortcuts = globalShortcuts,
        contentFactory = {
            ComposeUIViewController {
                var hardHidden by remember { mutableStateOf(false) }
                var showKeyboardShortcuts by remember { mutableStateOf(false) }
                var darkMode by remember { mutableStateOf(false) }
                MaterialTheme(colorScheme = if (darkMode) darkColorScheme() else lightColorScheme()) {
                    MainScreen(
                        listKeyboardShortcuts = globalShortcuts,
                        hardKeyboardHidden = hardHidden,
                        darkMode = darkMode,
                        showKeyboardShortcuts = { showKeyboardShortcuts = true },
                        toggleDarkMode = { darkMode = !darkMode }
                    )
                    KeyboardShortcuts(
                        enabled = showKeyboardShortcuts,
                        shortcuts = globalShortcuts
                    ) { showKeyboardShortcuts = false }
                }
            }
        }
    )
}
```

Compose content



Experts

```kotlin
fun MainViewController(): UIViewController {
    return KeyboardShortcutViewController(
        shortcuts = globalShortcuts,
        contentFactory = {
            ComposeUIViewController {
                var hardHidden by remember { mutableStateOf(false) }
                var showKeyboardShortcuts by remember { mutableStateOf(false) }
                var darkMode by remember { mutableStateOf(false) }
                MaterialTheme(colorScheme = if (darkMode) darkColorScheme() else lightColorScheme()) {
                    MainScreen(
                        listKeyboardShortcuts = globalShortcuts,
                        hardKeyboardHidden = hardHidden,
                        darkMode = darkMode,
                        showKeyboardShortcuts = { showKeyboardShortcuts = true },
                        toggleDarkMode = { darkMode = !darkMode }
                    )
                    KeyboardShortcuts(
                        enabled = showKeyboardShortcuts,
                        shortcuts = globalShortcuts
                    ) { showKeyboardShortcuts = false }
                }
            }
        }
    )
}
```

which in turn is passed to `KeyboardShortcutViewController`, along with the list of shortcuts

Compose content is passed to `ComposeUIViewController`

```swift
import UIKit
import SwiftUI
import ComposeApp

struct ComposeView: UIViewControllerRepresentable {
    func makeUIViewController(context: Context) -> UIViewController {
        MainViewControllerKt.MainViewController()
    }

    func updateUIViewController(_ uiViewController: UIViewController, context: Context) {
    }
}

struct ContentView: View {
    var body: some View {
        ComposeView()
            .ignoresSafeArea()
    }
}
```

which in turn is returned by `MainViewController`, which is called from Swift code

Experts

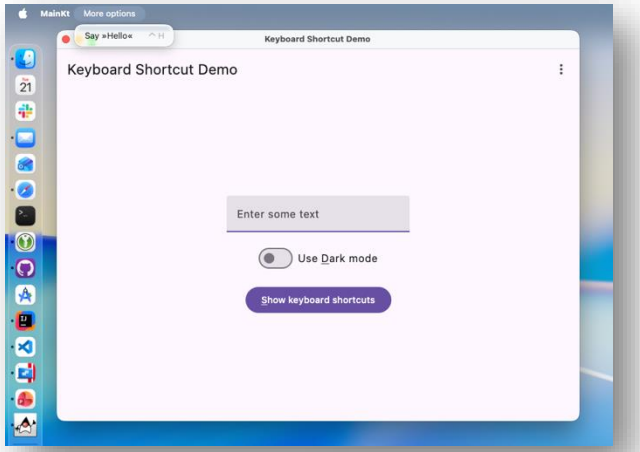
```kotlin
@OptIn(ExperimentalForeignApi::class, BetaInteropApi::class)
@ExportObjCClass
class KeyboardShortcutViewController(
    private val shortcuts: List<KeyboardShortcut>,
    private val contentFactory: () -> UIViewController,
) : UIViewController(nibName = null, bundle = null) {

    private lateinit var composeController: UIViewController

    override fun viewDidLoad() {
        super.viewDidLoad()
        composeController = contentFactory()
        addChildViewController(composeController)
        view.addSubview(composeController.view)
        composeController.view.autoresizingMask = UIViewAutoresizingFlexibleWidth or UIViewAutoresizingFlexibleHeight
        composeController.didMoveToParentViewController(this)
        shortcuts.forEach { shortcut ->
            var mask = 0L
            if (shortcut.ctrl) mask = mask or UIKeyModifierControl
            if (shortcut.meta) mask = mask or UIKeyModifierCommand
            if (shortcut.alt) mask = mask or UIKeyModifierAlternate
            if (shortcut.shift) mask = mask or UIKeyModifierShift
            UIKeyCommand.keyCommandWithInput(
                input = shortcut.keyAsString,
                modifierFlags = mask,
                action = NSSelectorFromString("handleCommand:")
            ).apply {
                discoverabilityTitle = shortcut.label
                addKeyCommand(this)
            }
        }
        becomeFirstResponder()
    }

    override fun viewDidLayoutSubviews() {
        super.viewDidLayoutSubviews()
        composeController.view.setFrame(view.bounds)
    }

    override fun canBecomeFirstResponder(): Boolean = true

    @ObjCAction
    fun handleCommand(sender: UIKeyCommand?) {
        shortcuts.find { it.shortcutAsText.endsWith(sender?.input ?: "") }?.triggerAction()
    }
}
```

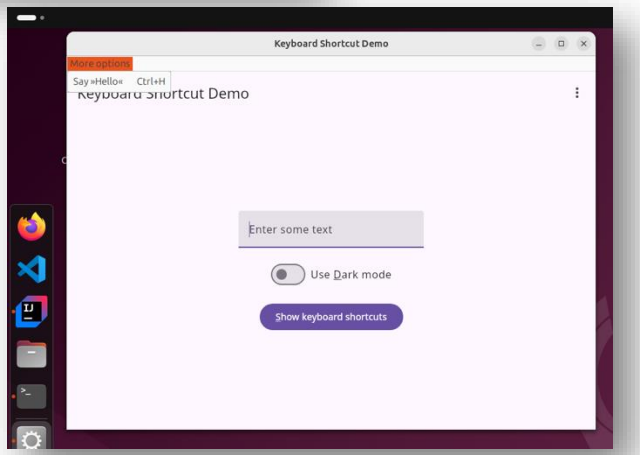
`KeyboardShortcutViewController` extends `UIViewController`

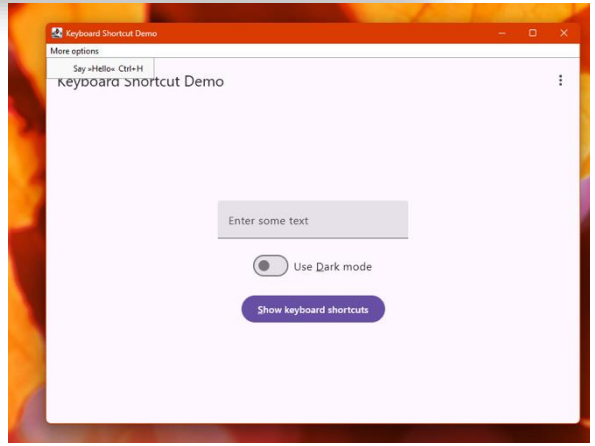
For each `KeyboardShortcut`, a `UIKeyCommand` is created and registered using `addKeyCommand()`

Resolve the function to be invoked upon a key press

Upon invocation, a matching instance of `KeyboardShortcut` is searched; if one is found, `triggerAction()` is invoked; for simplicity, we ignore modifier keys



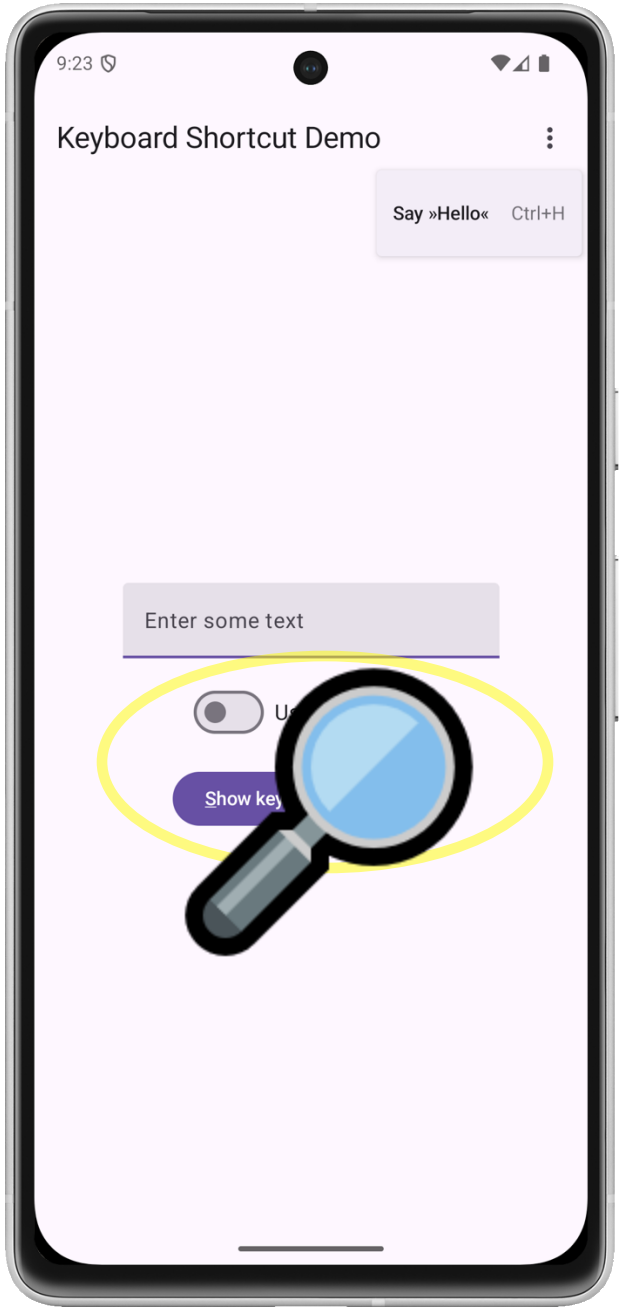


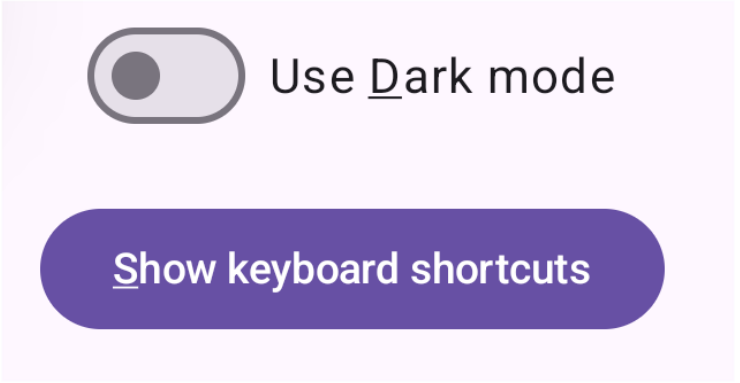


Experts

```kotlin
fun main() = application {
    val title = stringResource(Res.string.app_name)
    Window(
        onCloseRequest = ::exitApplication,
        title = title,
    ) {
        var showKeyboardShortcuts by remember { mutableStateOf(false) }
        var darkMode by remember { mutableStateOf(false) }
        MaterialTheme(colorScheme = if (darkMode) darkColorScheme() else lightColorScheme()) {
            MainScreen(
                listKeyboardShortcuts = globalShortcuts,
                hardKeyboardHidden = false,
                darkMode = darkMode,
                showKeyboardShortcuts = { showKeyboardShortcuts = true },
                toggleDarkMode = { darkMode = !darkMode }
            )
            MenuBar {
                Menu(text = stringResource(Res.string.more_options)) {
                    globalShortcuts.forEachIndexed { index, shortcut ->
                        Item(
                            text = shortcut.label,
                            shortcut = KeyShortcut(
                                shortcut.key,
                                shortcut.ctrl,
                                shortcut.meta,
                                shortcut.alt,
                                shortcut.shift,
                            ),
                            onClick = {
                                globalShortcuts[index].triggerAction()
                            })
                    }
                }
            }
            KeyboardShortcuts(
                enabled = showKeyboardShortcuts, shortcuts = globalShortcuts
            ) { showKeyboardShortcuts = false }
        }
    }
}
```

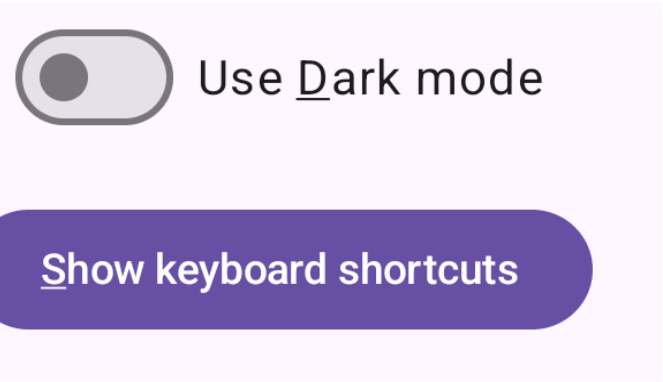
Access keys





1. Underline the key to be pressed with Alt
2. React to this key combination

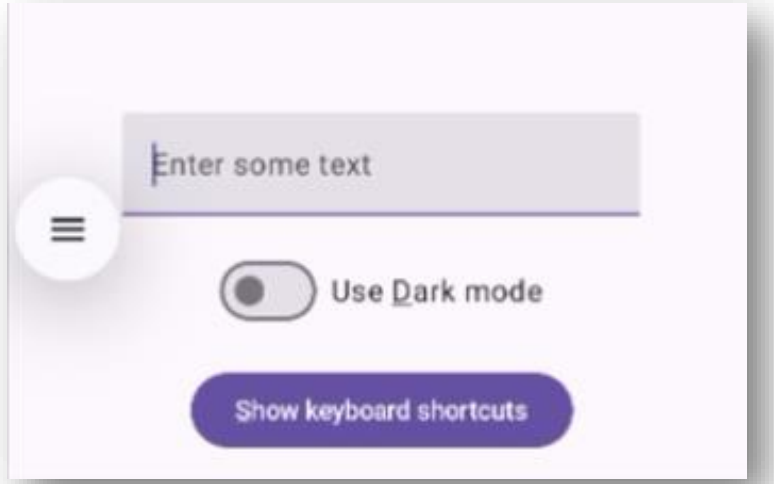
```
SwitchWithText(active = darkMode, text = {
    TextWithUnderlinedChar(stringResource(Res.string.dark_mode))
}) { toggleDarkMode() }
Button(
    onClick = showKeyboardShortcuts,
) {
    TextWithUnderlinedChar(stringResource(Res.string.show_keyboard_shortcuts))
}
```




```
@Composable
fun TextWithUnderlinedChar(text: String) {
    Text(buildAnnotatedString {
        val result = Regex("\\[([a-zA-Z])").find(text)
        val pos = result?.range?.first ?: -1
        if (pos >= 0) {
            append(text.substring(0, pos))
            result?.groupValues?.get(1)?.let { char ->
                withStyle(style = SpanStyle(textDecoration = TextDecoration.Underline)) {
                    append(char)
                }
            }
            append(text.substring(pos + 2))
        } else {
            append(text)
        }
    })
}
```

```
<string name="dark_mode">Use [Dark mode</string>
```

SnappMobile




```
1  TextField(
2      value = textFieldValue,
3      onValueChange = { textFieldValue = it },
4      modifier = Modifier.onKeyEvent {
5          if (it.type == KeyEventType.KeyUp && it.key == Key.Tab) {
6              scope.launch {
7                  animatable.animateTo(359f, animationSpec = tween(durationMillis = 1000))
8                  animatable.snapTo(0F)
9              }
10             true
11         } else {
12             false
13         }
14     }.graphicsLayer {
15         rotationY = animatable.value
16     },
17     placeholder = { Text(stringResource(Res.string.hint)) }
18 )
```

- When `onKeyEvent()` is invoked, the `TextField`'s default action for that keystroke has already been executed
- To receive `KeyEvent`s, the element must be focusable
- If the event has not been consumed (indicated by returning `false`), the parent element will be asked to handle it

`onPreviewKeyEvent()` modifier

- Intercepts key events before the component's default action is executed
- Events travel down the tree (parent-to-child) towards the currently focused component, allowing a parent to intercept events before its children
- Unlike `onKeyEvent()`, a composable using `onPreviewKeyEvent()` does not need to be focusable itself, as it can intercept events traveling down towards a focused child within its subtree
- A component within its subtree **must have focus**

```
@Composable
fun KeyboardShortcutDemo( … ) {
    …
    val focusRequester = remember { FocusRequester() }
    Scaffold(
        modifier = Modifier.fillMaxSize()
            .keyboardShortcuts(
                Pair(
                    Key.S, showKeyboardShortcuts
                ),
                Pair(
                    Key.D
                ) { toggleDarkMode() },
            )
            …,
        topBar = { … },
        snackbarHost = { SnackbarHost(snackBarHostState) }) { innerPadding ->
        Box(
            modifier = Modifier
                .focusRequester(focusRequester)
                …,
            contentAlignment = Alignment.Center
        ) {
            LaunchedEffect(focusRequester) {
                focusRequester.requestFocus()
            }
            Column( … ) { … }
            …
        }
        LaunchedEffect(snackbarMessage) { … }
        }
    }
}
```

```
/**
 * A modifier that listens for specific key presses.
 * Besides the specified key, Alt must also be pressed.
 *
 * @param shortcuts Pairs of keys and their corresponding actions
 */
expect fun Modifier.keyboardShortcuts(
    vararg shortcuts: Pair<Key, () -> Unit>
): Modifier
```

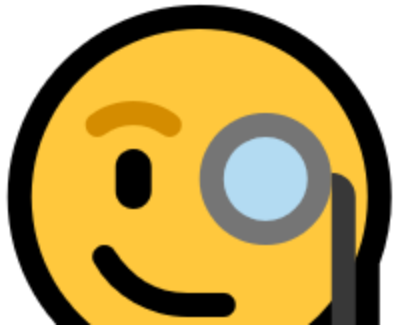
- Intercept key events as early (close to the root) as possible
- Have your content root request focus for best results

SnappMobile

```kotlin
// iOS
actual fun Modifier.keyboardShortcuts(
    vararg shortcuts: Pair<Key, () -> Unit>
): Modifier = then(
    Modifier.onPreviewKeyEvent { event ->
        if (event.type == KeyEventType.KeyDown &&
            event.isAltPressed) {
            shortcuts.forEach { (key, action) ->
                if (event.key == key) {
                    action()
                    true
                }
            }
        }
        event.type == KeyEventType.KeyDown
    }
)
```

```kotlin
// Desktop
actual fun Modifier.keyboardShortcuts(
    vararg shortcuts: Pair<Key, () -> Unit>
): Modifier = then(
    Modifier.onPreviewKeyEvent { event ->
        if (event.type == KeyEventType.KeyDown &&
            event.isAltPressed) {
            shortcuts.forEach { (key, action) ->
                if (event.key == key) {
                    action()
                    true
                }
            }
        }
        event.isAltPressed
    }
)
```

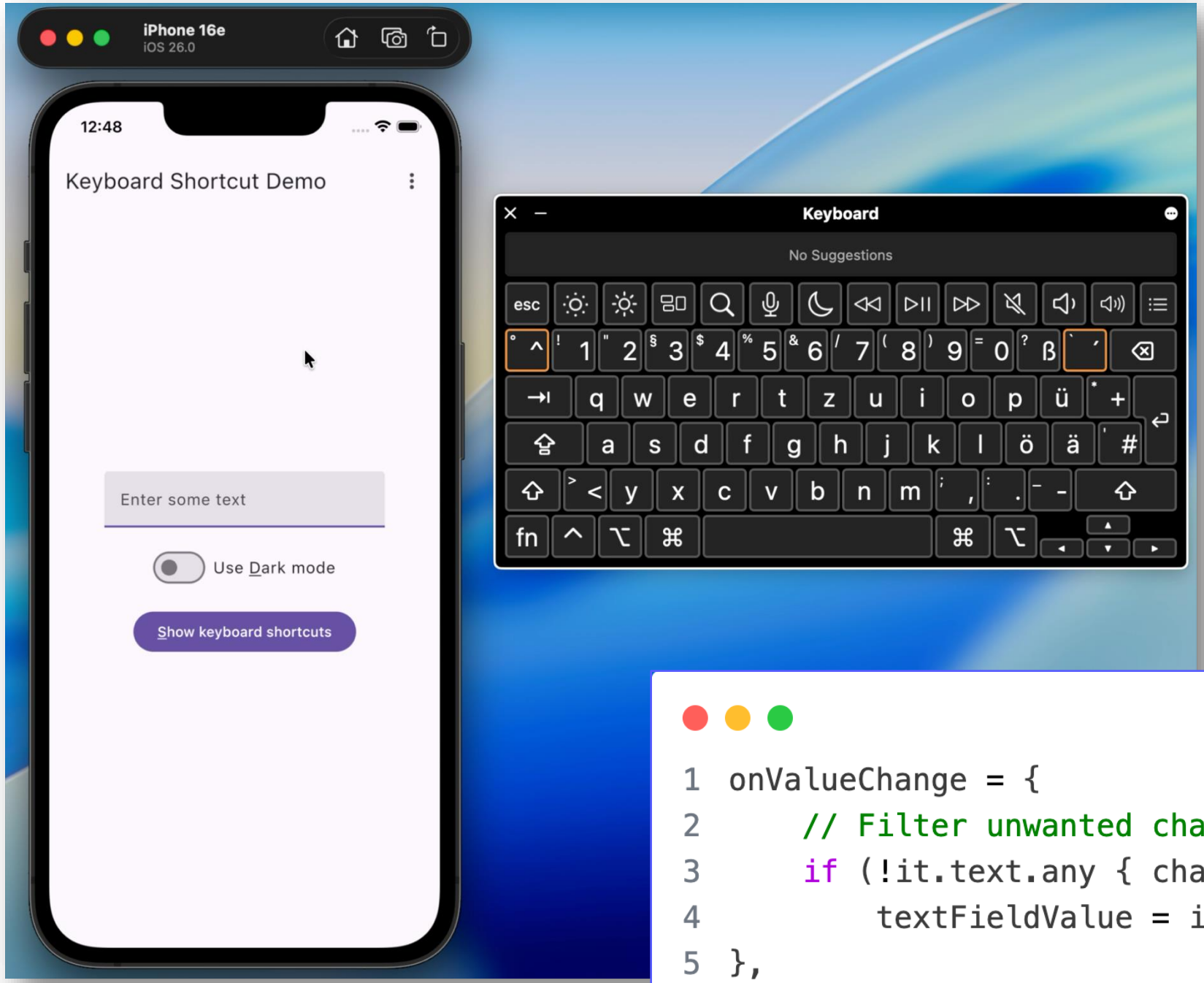
pretty much the same code



Experts

SnappMobile

- `onPreviewKeyEvent()` and `onKeyEvent()` report physical key states (`KeyEventType.KeyDown`, `KeyEventType.KeyUp`)
- no separate high-level *key typed* event
- Compose relies on the native event handling of each operating system, which produce different `KeyEventType` sequences
- On iOS, returning `true` may not always stop an event from propagating



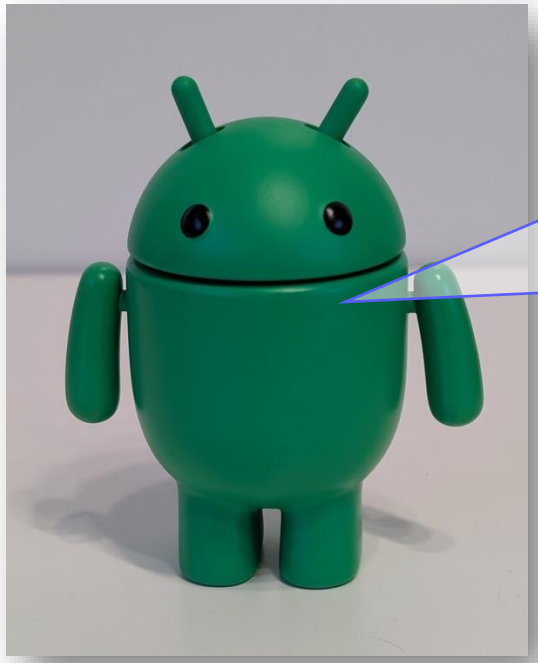

```
onValueChange = {
    // Filter unwanted chars on iOS
    if (!it.text.any { char -> char in listOf('ə', ',') })
        textFieldValue = it
},
```

```android
// Android
actual fun Modifier.keyboardShortcuts(
    vararg shortcuts: Pair<Key, () -> Unit>
): Modifier = then(
    Modifier.onInterceptKeyBeforeSoftKeyboard() { event ->
        if (event.type == KeyEventType.KeyDown) {
            shortcuts.forEach { (key, action) ->
                if (event.key == key && event.isAltPressed) {
                    action()
                    true
                }
            }
        }
        event.isAltPressed
    }
)
```

Makes sure that Gboard doesn't interfere

```desktop
// Desktop
actual fun Modifier.keyboardShortcuts(
    vararg shortcuts: Pair<Key, () -> Unit>
): Modifier = then(
    Modifier.onPreviewKeyEvent { event ->
        if (event.type == KeyEventType.KeyDown &&
            event.isAltPressed) {
            shortcuts.forEach { (key, action) ->
                if (event.key == key) {
                    action()
                    true
                }
            }
        }
        event.isAltPressed
    }
)
```

```
1  Scaffold(
2      modifier = Modifier.fillMaxSize()
3          .keyboardShortcuts( … )
4          // Depending on the platform, global shortcuts may not be delivered when a composable
5          // has requested focus, so we make sure to handle global shortcuts here, too
6          .onPreviewKeyEvent { event ->
7              shortcuts.forEach { shortcut ->
8                  if (shortcut.key == event.key &&
9                      shortcut.ctrl == event.isCtrlPressed &&
10                     shortcut.meta == event.isMetaPressed &&
11                     shortcut.alt == event.isAltPressed &&
12                     shortcut.shift == event.isShiftPressed
13                 ) {
14                     shortcut.triggerAction()
15                     return@onPreviewKeyEvent true
16                 }
17             }
18             false
19         },
20     topBar = {
21         …
```

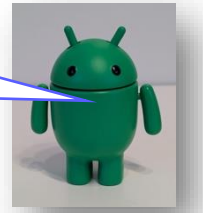


That’s been amazing, right?

There’s one more thing!

```kotlin
class MainActivity : ComponentActivity() {
    …
    private var hardKeyboardHiddenFlow = MutableStateFlow(-1)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate
        …
        updateFlows()
        setContent {
            val hardKeyboardHidden by hardKeyboardHiddenFlow.collectAsStateWithLifecycle()
            MaterialTheme(colorScheme = if (darkMode) darkColorScheme() else lightColorScheme()) {
                MainScreen(
                    listKeyboardShortcuts = globalShortcuts,
                    hardKeyboardHidden = hardKeyboardHidden == Configuration.HARDKEYBOARDHIDDEN_YES,
                    …
                )
            }
        }
    }

    …

    override fun onConfigurationChanged(newConfig: Configuration) {
        super.onConfigurationChanged(newConfig)
        updateFlows()
    }

    private fun updateFlows() {
        hardKeyboardHiddenFlow.update {
            resources.configuration.hardKeyboardHidden
        }
    }
}
```

Detecting physical keyboards on Android



Gets updated when a keybnoard is available / not available

```xml
<activity
    android:exported="true"
    android:configChanges="keyboard|keyboardHidden"
    android:name=".MainActivity">
```

read from `resources.configuration.hardwareKeyboardHidden`

Key Takeaways

- Shortcuts are essential for expert users and fast-paced interaction
- Physical keyboards are becoming more common on Android (ChromeOS, Desktop Mode)
- A few UI elements have built-in shortcut support

Keyboard shortcut	Action	Composables supporting the shortcut
`Shift+Ctrl+Left arrow/Right arrow`	Select text to beginning/end of word	`BasicTextField`, `TextField`
`Shift+Ctrl+Up arrow/Down arrow`	Select text to beginning/end of paragraph	`BasicTextField`, `TextField`
`Shift+Alt+Up arrow/Down arrow` or `Shift+Meta+Left arrow/Right arrow`	Select text to beginning/end of text	`BasicTextField`, `TextField`
`Shift+Left arrow/Right arrow`	Select characters	`BasicTextField`, `TextField`
`Ctrl+A`	Select all	`BasicTextField`, `TextField`
`Ctrl+C/Ctrl+X/Ctrl+V`	Copy/cut/paste	`BasicTextField`, `TextField`
`Ctrl+Z/Ctrl+Shift+Z`	Undo/redo	`BasicTextField`, `TextField`
`PageDown/PageUp`	Scroll	`LazyColumn`, the `verticalScroll` modifier, the `scrollable` modifier

https://developer.android.com/develop/ui/compose/touch-input/keyboard-input/commands

Challenges Ahead

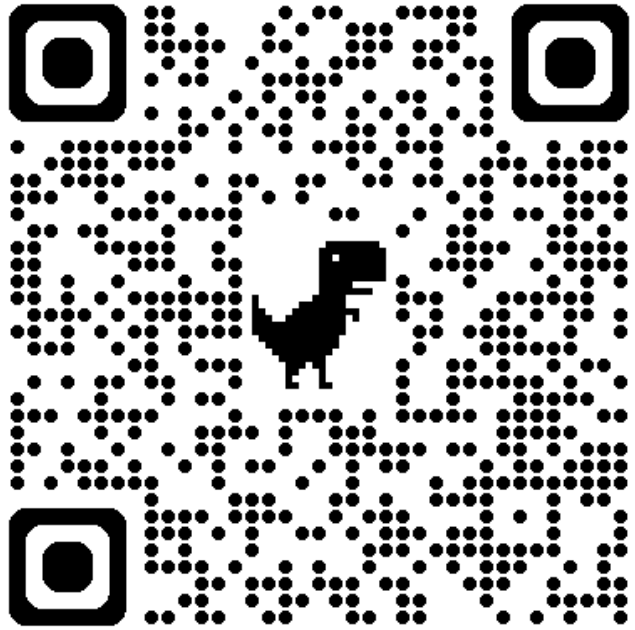
- Today's Compose APIs (`onKeyEvent()`, `onPreviewKeyEvent()`) cover basic shortcut requirements
- There’s no reliable *key typed* event
- There is no easy way to get human-readable text for key combinations

Thank you!

@tkuenneth

@tkuenneth

@tkuenneth.dev



https://github.com/tkuenneth/KeyboardShortcutDemo

Experts

SnappMobile