Desktop Applications

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346

Contents

Introduction

Graphical User Interfaces

The term Graphical User Interface refers to any interface that primarily uses a drawing surface to draw and animate objects, and supports primary interaction through that surface.

Features include:

- Graphical environment for text/graphics output.
- Interactive graphical elements e.g., buttons, menus, lists.
- Rich media support e.g., animations, graphics, sound.
- Pointing device (“point-and-click interaction”).

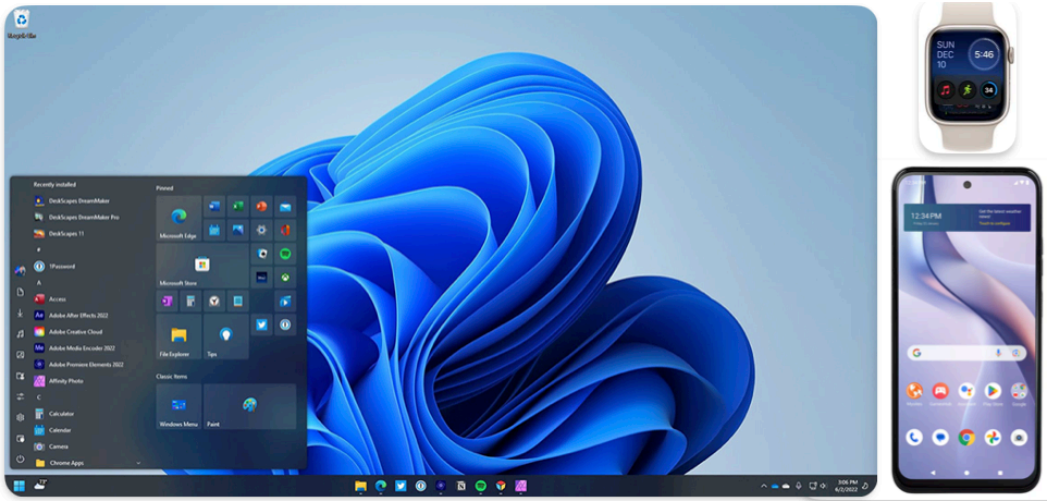


Figure 1: Desktop, mobile devices all offer graphical user interfaces.

Desktop Interfaces

A desktop is a specific form of GUI that originated at Xerox PARC in 1973. It wasn't popularized until the release of the Apple Macintosh in 1984.

In addition to GUI features, desktop interfaces follow the WIMP pattern:

- **Window** contain each each program, and provide security/protection.
- **Icons** represent interactive objects and support interaction.
- **Menus** represent commands that can be issued by the user.
- **Pointer** refers to a cursor, manipulated by a pointing device.

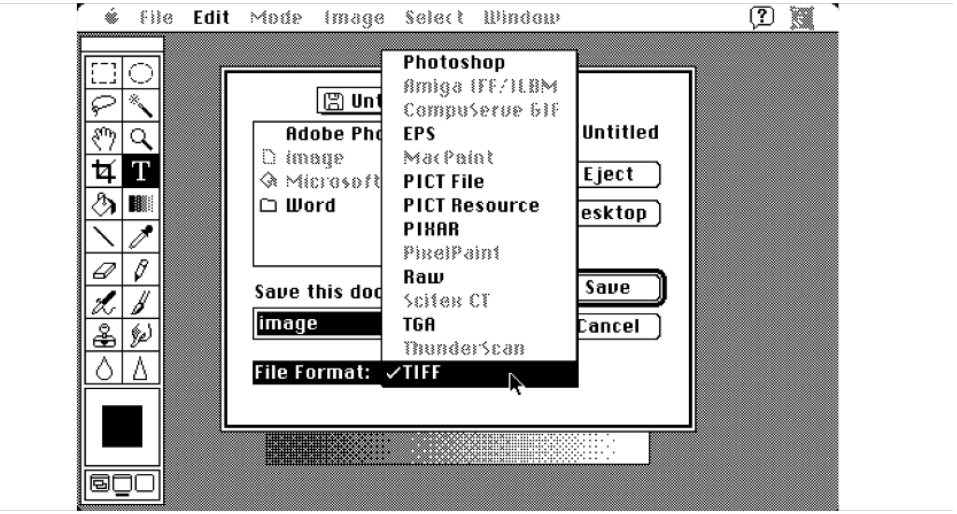


Figure 2: The Apple Macintosh offered the first commercially successful GUI. Many other companies followed this design closely, including MS Windows.

Using Compose

Compose Multiplatform is the main GUI toolkit for building desktop applications.[1]

All of the composables and concepts that we covered in the User Interface lecture apply here! We just want to highlight some of the features that are specific to desktop application (and which might not appy to Web or Mobile applications).

A desktop application always includes

- one or more windows, where one of them is “active”.
- interactive window contents, with standard components for input/output.
- drop-down menus, and context-menus.
- support for text and keyboards.

Installation

To install the Compose libraries and make then accessible, you need to update your `module.yaml` file:

module.yaml

```
product: jvm/app

dependencies:
  - $compose.desktop.currentOs

settings:
  compose: enabled
```

Rebuild to import the dependencies, and you can start using Compose.

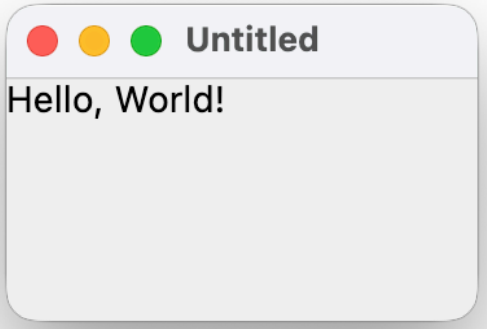
Composables

Window

A window is the top-level of our scene-graph.

- Windows are “owned” by the application. A desktop GUI will direct all output to this window.
- There is always at least one window and there may be multiples.
- Compose handles regular behaviour e.g., resizing, minimize, maximize.

```
fun main() = application {
  Window(onCloseRequest = ::exitApplication) {
    BasicText("Hello, World!")
  }
}
```

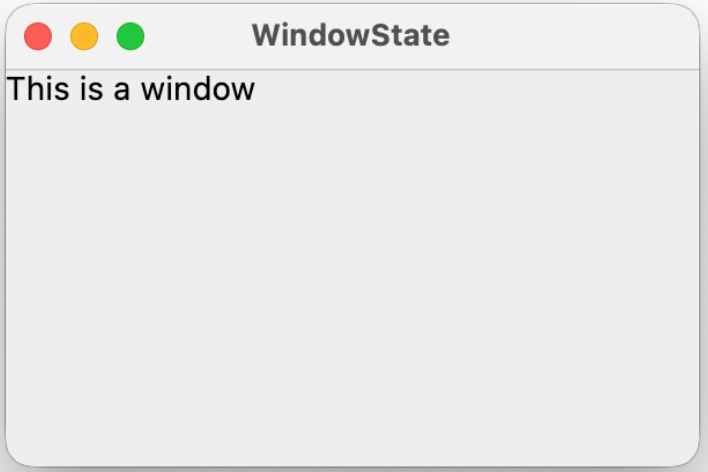


Window

WindowState

Window size and position is stored in a `WindowState` object and passed as an argument into the window.

```
fun main() {
  application {
    Window(
      title = "WindowState",
      state = WindowState( // state obj manages size and position
        position = WindowPosition(Alignment.center),
        size = DpSize(300.dp, 200.dp)
      ),
      onCloseRequest = ::exitApplication
    ) {
      Text("This is a window")
    }
  }
}
```

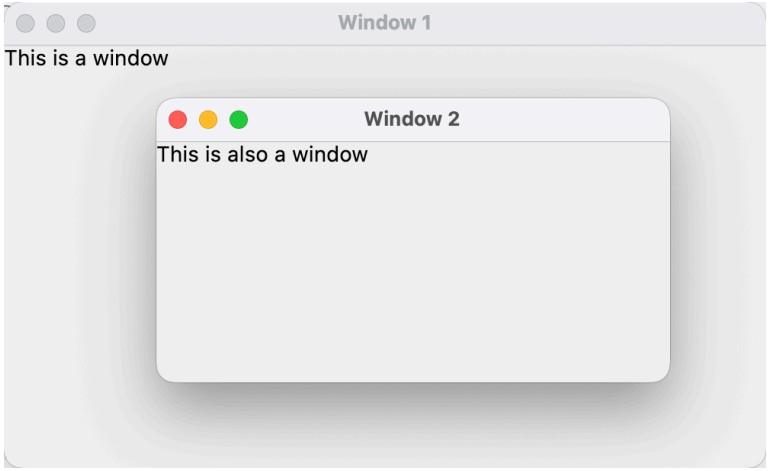


Multiple Windows

Multiple windows are easy to implement. Define each window within the same `application` context.

```
fun main() {
  application {
    Window(
      title = "Window 1",
      onCloseRequest = ::exitApplication
    ) {
      Text("This is a window")
    }

    Window(
      title = "Window 2",
      onCloseRequest = ::exitApplication
      ) {
        Text("This is also a window")
      }
  }
}
```



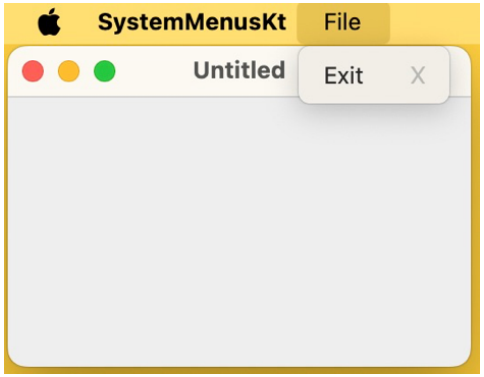
Dialog Box

A dialog is a special type of floating window that sits in the foreground.

```
Window(
  title = "Main Window", onCloseRequest = ::exitApplication,
  state = WindowState(position = WindowPosition(Alignment.Center))
  ){
    var isDialogOpen by remember { mutableStateOf(false) }
    Button(onClick = { isDialogOpen = true }) {
      Text(text = "Open dialog")
  }
  if (isDialogOpen) { // flag determines if this gets shown
    DialogWindow(
      title = "Dialog", onCloseRequest = { isDialogOpen = false },
      state = rememberDialogState(
        position = WindowPosition(Alignment.Center))
    ){
      Text("Dialog text goes here")
    }
  }
}
```

System Menus

Menus attached to the window/titlebar.

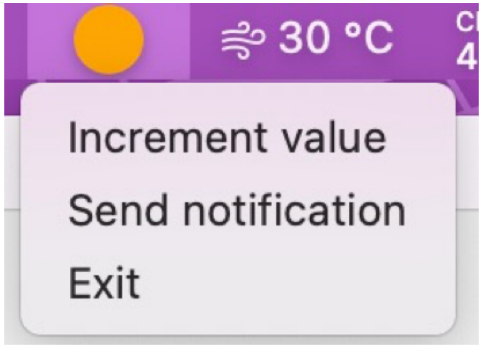



```
fun main() = application {
  Window(onCloseRequest = ::exitApplication) {
    App(this@Window, this@application)
  }
}

@Composable
fun App(windowScope: FrameWindowScope, appScope: ApplicationScope) {
  windowScope.MenuBar {
    Menu("File", mnemonic = 'F') {
      val nextWindowState = rememberWindowState()
      Item("Exit",
        onClick = { appScope.exitApplication() },
        shortcut = KeyShortcut(Key.X, ctrl = false)
      )
    }
  }
}
```

System Tray

Notification area.




```
val trayState = rememberTrayState()
val notification = rememberNotification(
  "Notification", "Message from MyApp!"
)

Tray(
  state = trayState, icon = TrayIcon,
  menu = {
    Item("Increment value",
      onClick = { count++}
    )
    Item("Send notification",
      onClick = { trayState.sendNotification(notification)}
    )
    Item("Exit",
      onClick = { isOpen = false }
    )
  })
```

Interaction

Keyboard Input

Keyboard input relies on event handlers attached to a window that can capture specific keystrokes.

```
fun main() = application {
  Window(
      title = "Key Events",
      state = WindowState(width = 500.dp, height = 100.dp),
      onCloseRequest = ::exitApplication,
      onKeyEvent = { // window-level
        if (it.type == KeyEventType.KeyUp) { println(it.key) }
      }
  ){
    val text = remember { mutableStateOf("") }
    val textField = TextField(
      value = text.value,
      onValueChange = { text.value = it } // widget level
    )
  }
}
```

Mouse Clicks

Mouse input is typically captured by a Modifier property called `combinedClickable`.

```
Box(
  modifier = Modifier
    .background(Color.Magenta)
    .fillMaxWidth(0.9f)
    .fillMaxHeight(0.2f)
    .combinedClickable(
      onClick = { text = "Click! ${count++}" },
      onDoubleClick = { text = "Double click! ${count++}" },
      onLongClick = { text = "Long click! ${count++}" }
    )
)
```

Mouse Drag

Mouse drag input is handled by tracking pointer movement.

```
var color by remember { mutableStateOf(Color(0, 0, 0)) }

Box(
  modifier = Modifier
    .background(Color.Magenta)
    .fillMaxWidth(0.9f)
    .fillMaxHeight(0.2f)
    .onPointerEvent(PointerEventType.Move) {
        val position = it.changes.first().position
        color = Color(
          position.x.toInt() % 256,
          position.y.toInt() % 256, 0
        )
    }
)
```

Bibliography

[1] JetBrains Inc., “Compose Multiplatform.” [Online]. Available: https://kotlinlang.org/compose-multiplatform/