Declarative UI for B4A

Forum Guide and Quick API Tour

Version 0.1 demo | Contract version 1.0

1. What this project is

Declarative UI for B4A is a small, code-first composition layer for native B4A applications. It lets an application describe a tree of widgets and containers while the library handles native view creation, measurement, positioning and rendering.

It is inspired by Flutter, SwiftUI and Jetpack Compose, but it is not a Flutter clone and it does not introduce a new programming language. The syntax is normal B4A with optional fluent method chaining:

```
Dim title As UILabel
title.Initialize _
.Text("Today’s tasks") _
.Size(24) _
.Color(0xFF132238)

Dim body As UIColumn
body.Initialize _
.Spacing(12dip) _
.AddChild(title) _
.AddChild(actionButton)
```

The result is still a native B4A UI. The library provides composition and layout conventions; it does not hide the Android platform or pretend that a virtual screen is a physical Activity.

2. Current scope

This is an early B4A-only development/demo release. It currently provides:

- declarative containers: UIColumn, UIRow, UIStack, UIPadding, UIBox, UICard, UICenter and UIExpanded;
- native controls: UILabel, UIButton, UIFloatingActionButton, UIInput, UIIcon, UIImage and UIProgressBar;
- overlays and feedback: UIAlertDialog, UISnackBar and UIPlaceholder;
- app structure: UIAppBar, UIScaffold, UINavigator and UIBottomNavigationBar;
- layout helpers: UIVisibility, UIScrollView and UIListView;
- state and behavior: UIState, UIAsyncState and UIAnimation;
- design tokens: UITheme with light/dark schemes and seed colors;
- native interoperation through UINative.

The project does not currently promise complete Flutter compatibility, automatic virtual-DOM diffing, automatic two-way binding, variable-height virtualization or cross-platform B4X behavior.

3. Available widgets and building blocks

The following catalog describes the complete set of classes included in the current demo library. Methods listed here are the application-facing API; SetParent, SetPosition, SetSize, Render, Unmount and GetContentSize are the shared composition protocol and are normally called by parent containers.

Layout and composition widgets

Widget Purpose Main methods
--- --- ---
‘UIColumn‘ Arranges children vertically and measures their natural heights. ‘AddChild‘, ‘Spacing‘, ‘MainAxisSize‘, ‘MainAxisAlignment‘, ‘CrossAxisAlignment‘, ‘ApplyTheme‘

‘UIRow‘ Arranges children horizontally and measures their natural widths. ‘AddChild‘, ‘Spacing‘, ‘MainAxisSize‘, ‘MainAxisAlignment‘, ‘CrossAxisAlignment‘, ‘ApplyTheme‘
‘UIPadding‘ Wraps one child with configurable insets. ‘All‘, ‘Horizontal‘, ‘Vertical‘, ‘Only‘, ‘Child‘, ‘ApplyTheme‘
‘UIBox‘ Lightweight padded wrapper for one child. ‘Initialize(child, padding)‘, ‘ApplyTheme‘
‘UICard‘ Rounded surface with optional background and border. ‘Child‘, ‘BackgroundColor‘, ‘BorderColor‘, ‘CornerRadius‘, ‘ApplyTheme‘, ‘GetView‘
‘UICenter‘ Centers one child using its measured natural size. ‘Child‘, ‘ApplyTheme‘
‘UIExpanded‘ Marks a child as flexible so it receives remaining main-axis space. ‘Child‘, ‘GetChild‘, ‘ApplyTheme‘
‘UIStack‘ Places children on the Z axis; later children appear above earlier ones. ‘AddChild‘, ‘Alignment‘, ‘ApplyTheme‘
‘UIVisibility‘ Includes or removes one child from layout without losing its declarative configuration. ‘Visible‘, ‘BindVisible‘, ‘UnbindVisible‘, ‘OnVisibilityChanged‘, ‘Child‘, ‘ApplyTheme‘
‘UISpace‘ Adds a fixed empty gap in a layout. ‘Size‘

A compact composition example:

```
Dim title As UILabel
title.Initialize.Text("Daily routine").Size(22)

Dim actions As UIRow
actions.Initialize _
.Spacing(8dip) _
.AddChild(saveButton) _
.AddChild(cancelButton)

Dim column As UIColumn
column.Initialize _
.Spacing(12dip) _
.AddChild(title) _
.AddChild(actions)

Dim cardPadding As UIPadding
cardPadding.Initialize.All(16dip).Child(column)

Dim card As UICard
card.Initialize.Child(cardPadding)
```

Use UIExpanded when a child should fill the remaining space:

```
Dim expandedScroll As UIExpanded
expandedScroll.Initialize.Child(scroll)

Dim body As UIColumn
body.Initialize _
.AddChild(header) _
.AddChild(expandedScroll) _
.AddChild(footer)
```

Use UIStack for badges and overlays:

```
Dim stack As UIStack
stack.Initialize _
.Alignment("bottomRight") _
.AddChild(backgroundCard) _
.AddChild(statusBadge)
```

Text, input and action widgets

Widget Purpose Main methods
--- --- ---
‘UILabel‘ Native text label with natural measurement. ‘Text‘, ‘BindText‘, ‘UnbindText‘, ‘Size‘, ‘Color‘, ‘ApplyTheme‘
‘UIButton‘ Native clickable button. ‘Text‘, ‘BindText‘, ‘UnbindText‘, ‘BackgroundColor‘, ‘TextColor‘, ‘TextSize‘, ‘CornerRadius‘, ‘Border‘, ‘OnClick‘, ‘TriggerClick‘

‘UIFloatingActionButton‘ Compact floating action button for a primary screen action. ‘Text‘, ‘BindText‘, ‘UnbindText‘, ‘BackgroundColor‘, ‘TextColor‘, ‘TextSize‘, ‘CornerRadius‘, ‘OnClick‘, ‘ApplyTheme‘
‘UIInput‘ Native B4A ‘EditText‘ inside the declarative layout protocol. ‘Hint‘, ‘Text‘, ‘BindText‘, ‘UnbindText‘, ‘OnTextChanged‘, ‘TextColor‘, ‘HintColor‘, ‘BackgroundColor‘, ‘TextSize‘, ‘CornerRadius‘, ‘Border‘, ‘GetText‘
‘UIIcon‘ Material Icons, FontAwesome or regular Unicode glyph. ‘Material‘, ‘FontAwesome‘, ‘Unicode‘, ‘MaterialCode‘, ‘FontAwesomeCode‘, ‘Size‘, ‘Color‘, ‘Alignment‘, ‘OnClick‘
‘UIImage‘ Image from Files or a public URL with placeholder support. ‘Asset‘, ‘Network‘, ‘PlaceholderAsset‘, ‘Fit‘, ‘Width‘, ‘Height‘, ‘OnLoaded‘, ‘OnError‘, ‘ApplyTheme‘
‘UIPlaceholder‘ Lightweight Flutter-style placeholder box for loading or reserved content. ‘Color‘, ‘StrokeWidth‘, ‘Width‘, ‘Height‘, ‘Size‘, ‘ApplyTheme‘
‘UIProgressBar‘ Determinate or indeterminate progress indicator. ‘Value‘, ‘BindValue‘, ‘UnbindValue‘, ‘Indeterminate‘, ‘Height‘, ‘ApplyTheme‘
‘UIDivider‘ Themed horizontal separator. ‘Color‘, ‘ApplyTheme‘

Example using common controls:

```
Dim name As UIInput
name.Initialize _
.Hint("Your name") _
.CornerRadius(10dip) _
.OnTextChanged(Me, "Name_Changed")

Dim submit As UIButton
submit.Initialize _
.Text("CONTINUE") _
.OnClick(Me, "Continue_Click")

Dim progress As UIProgressBar
progress.Initialize.Value(65)

Dim icon As UIIcon
icon.Initialize.FontAwesomeCode(0xF013).Size(22)
```

UIIcon.MaterialCode and UIIcon.FontAwesomeCode accept integer code points, so code such as FontAwesomeCode(0xF186) is supported directly. UIImage.Network is intended for public URLs; the host remains responsible for network permissions and for handling failures through OnError or PlaceholderAsset.

Screen structure and navigation widgets

Widget Purpose Main methods
--- --- ---
‘UIAppBar‘ Top bar with title and optional action widget. ‘Title‘, ‘BindTitle‘, ‘UnbindTitle‘, ‘Action‘, ‘ClearAction‘, ‘BackgroundColor‘, ‘TitleColor‘, ‘TitleSize‘, ‘ApplyTheme‘
‘UIScaffold‘ Screen shell that reserves space for app bar, body, bottom navigation and FABs. ‘AppBar‘, ‘Body‘, ‘BottomNavigationBar‘, ‘FloatingActionButtonLeft‘, ‘FloatingActionButtonRight‘, ‘BackgroundColor‘, ‘ApplyTheme‘
‘UINavigator‘ Virtual screen registration and single-Activity navigation with safe-area handling. ‘AddScreen‘, ‘NavigateTo‘, ‘ApplyTheme‘, ‘RefreshInsets‘
‘UIBottomNavigationBar‘ Data-driven bottom navigation with selected state and callbacks. ‘AddItem‘, ‘BindSelectedIndex‘, ‘OnSelected‘, ‘SetSelectedIndex‘, ‘GetSelectedIndex‘, ‘GetSelectedId‘, ‘ShowInactiveLabels‘, ‘ApplyTheme‘

A screen shell can be assembled as follows:

```
Dim bar As UIAppBar
bar.Initialize.Title("Home").Action(themeIcon)

Dim screen As UIScaffold
screen.Initialize _
.AppBar(bar) _
.Body(body) _
.FloatingActionButtonRight(addButton)

Navigator.Initialize _
```

```
.AddScreen("Home", screen) _
.AddScreen("Settings", settingsScreen)
Navigator.SetParent(Activity)
Navigator.SetPosition(0, 0)
Navigator.SetSize(Activity.Width, Activity.Height)
Navigator.NavigateTo("Home")
Navigator.Render
```

Bottom navigation items are data rather than separate native screens:

```
Dim tabs As UIBottomNavigationBar
tabs.Initialize _
.AddItem("home", "?", "Home") _
.AddItem("settings", "?", "Settings") _
.OnSelected(Me, "Tab_Selected")

Sub Tab_Selected(Index As Int, Id As String)
Navigator.NavigateTo(Id)
End Sub
```

Overlay and feedback widgets

Widget Purpose Main methods
--- --- ---
‘UISnackBar‘ Temporary message overlay with an optional action. ‘Message‘, ‘Action‘, ‘Duration‘, ‘AnimationDuration‘, ‘BackgroundColor‘, ‘TextColor‘, ‘ActionColor‘, ‘CornerRadius‘, ‘Margin‘, ‘Show‘, ‘Dismiss‘
‘UIAlertDialog‘ Modal overlay with title, message, optional declarative content and positive/negative actions. ‘Title‘, ‘Message‘, ‘Content‘, ‘PositiveButton‘, ‘NegativeButton‘, ‘DismissOnOutside‘, ‘ButtonColor‘, ‘ButtonTextColor‘, ‘CornerRadius‘, ‘ApplyTheme‘, ‘Show‘, ‘Dismiss‘
‘UIAnimation‘ Explicit bounds animation for an already-mounted native view. ‘TargetView‘, ‘MoveTo‘, ‘SizeTo‘, ‘MoveAndResize‘, ‘Duration‘, ‘OnCompleted‘, ‘Start‘, ‘Cancel‘, ‘IsRunning‘

Example dialog and feedback flow:

```
Dim dialog As UIAlertDialog
dialog.Initialize _
.Title("Delete item?") _
.Message("This action cannot be undone.") _
.NegativeButton("Cancel", Me, "DialogCancel") _
.PositiveButton("Delete", Me, "DialogDelete") _
.ApplyTheme(AppTheme) _
.Show(Activity)

Dim snack As UISnackBar
snack.Initialize.Message("Item deleted").Duration(2500).Show(Activity)
```

Scrolling, lists and native interoperability

Widget Purpose Main methods
--- --- ---
‘UIScrollView‘ Native vertical ScrollView hosting one declarative child. ‘Child‘, ‘GetChild‘, ‘ScrollTo‘, ‘GetScrollPosition‘, ‘ApplyTheme‘
‘UIListView‘ Fixed-height virtualized list with pooled declarative rows. ‘Items‘, ‘ItemCount‘, ‘ItemHeight‘, ‘Overscan‘, ‘CreateItem‘, ‘BindItem‘, ‘NotifyDataSetChanged‘, ‘GetItem‘, ‘ApplyTheme‘
‘UINative‘ Adapter that inserts an existing native B4A view into a declarative container. ‘Initialize(nativeView, naturalWidth, naturalHeight)‘, ‘GetView‘, ‘ApplyTheme‘

UIScrollView is suitable for small or variable-height content. UIListView is suitable for long collections with a fixed row height. UINative keeps ownership and event handling with the host Activity:

```
Dim nativeButton As Button
nativeButton.Initialize("NativeButton")

Dim nativeWidget As UINative
```

```
nativeWidget.Initialize(nativeButton, 140dip, 48dip)

Dim row As UIRow
row.Initialize.AddChild(nativeWidget).AddChild(saveButton)
```

State and design-token classes

These are not visual widgets, but they are part of the public building-block set:

Class Purpose Main methods
--- --- ---
‘UIState‘ Observable replacement-value state for labels, inputs, visibility and custom callbacks. ‘Initialize‘, ‘GetState‘, ‘SetState‘, ‘Subscribe‘, ‘Unsubscribe‘, ‘UnsubscribeTarget‘, ‘ClearListeners‘
‘UIAsyncState‘ Idle/loading/success/error snapshot for operations that complete later. ‘Initialize‘, ‘SetIdle‘, ‘SetLoading‘, ‘SetSuccess‘, ‘SetError‘, ‘Reset‘, ‘GetStatus‘, ‘GetValue‘, ‘GetErrorMessage‘, ‘Subscribe‘
‘UITheme‘ Material-like semantic colors, typography, shapes and layout tokens. ‘Initialize‘, ‘InitializeDark‘, ‘InitializeWithScheme‘, ‘InitializeWithSchemeAndMode‘, ‘Scheme‘, ‘Toggle‘, ‘DarkMode‘

UIAsyncState does not perform HTTP requests. It exposes the operation state while the host continues to use normal B4A Wait For, ResumableSub, HttpJob, database or file code.

4. Installation

1. Copy ‘DeclarativeUI.b4xlib‘ to the B4A Additional Libraries folder.
2. Refresh the Libraries Manager in the B4A IDE.
3. Enable the ‘DeclarativeUI‘ library.
4. Enable the required B4A libraries used by the host project: ‘XUI‘ and ‘IME‘.
5. Create a normal B4A Activity project and use the UI classes from the library.

4. The basic lifecycle

A layout-aware widget follows this lifecycle:

```
Initialize -> configure -> compose -> mount -> Render
|
Unmount
```

The public composition protocol is:

```
SetParent(parent As B4XView)
SetPosition(left As Int, top As Int)
SetSize(width As Int, height As Int)
Render
Unmount
GetContentSize(maxWidth As Int, maxHeight As Int) As List
```

Application code normally configures a root tree and renders only that root. Containers call the protocol on their children. Custom widgets can participate when they implement the same methods.

A direct root example:

```
Sub Activity_Create(FirstTime As Boolean)
Dim root As B4XView = Activity
root.Color = 0xFFF4F7FB

Dim title As UILabel
title.Initialize.Text("Hello declarative UI").Size(24).Color(0xFF132238)

title.SetParent(root)
title.SetPosition(24dip, 40dip)
title.SetSize(root.Width - 48dip, 48dip)
title.Render
End Sub
```

For real screens, use a container instead of manually positioning every child.

5. Composition and natural layout

UIColumn lays out children vertically. UIRow lays them out horizontally. UIPadding adds insets, UICard provides a rounded surface, and UIExpanded consumes remaining space.

```
Dim heading As UILabel
heading.Initialize.Text("Daily routine").Size(22)

Dim save As UIButton
save.Initialize.Text("SAVE").OnClick(Me, "Save_Click")

Dim content As UIColumn
content.Initialize _
.Spacing(10dip) _
.CrossAxisAlignment("stretch") _
.AddChild(heading) _
.AddChild(save)

Dim screenBody As UIPadding
screenBody.Initialize.All(16dip).Child(content)
```

Useful layout options are:

```
column.MainAxisSize("min")
column.MainAxisAlignment("center")
column.CrossAxisAlignment("stretch")
row.MainAxisAlignment("spaceBetween")
```

Supported main-axis alignment values are start, center, end, spaceBetween, spaceAround and spaceEvenly. Cross-axis values are stretch, start, center and end. Values are case-insensitive.

A child reports its natural size through GetContentSize, returning List(width, height). An empty list means flexible. UIColumn and UIRow measure natural children first and give remaining space to UIExpanded children. This is why text and cards can grow naturally instead of depending on arbitrary fixed heights.

A scrollable body normally uses UIExpanded:

```
Dim scroll As UIScrollView
scroll.Initialize.Child(content)

Dim expandedScroll As UIExpanded
expandedScroll.Initialize.Child(scroll)

Dim body As UIColumn
body.Initialize _
.AddChild(header) _
.AddChild(expandedScroll) _
.AddChild(footer)
```

6. State-driven UI

UIState is an explicit observable value holder. It is intentionally small: it does not replace the application model and it does not observe mutations inside an existing Map or List.

```
Private CounterState As UIState
Private CounterLabel As UILabel

CounterState.Initialize(0)
CounterLabel.Initialize.BindText(CounterState).Size(52)

Sub Increment_Click
Dim currentValue As Int = CounterState.GetState
CounterState.SetState(currentValue + 1)
End Sub
```

BindText applies the current value immediately and updates the label when the state changes. Text(...) replaces the binding and makes the text static. The same pattern is available on UIButton, UIFloatingActionButton, UIInput and UIAppBar through BindText or BindTitle.

For custom reactions:

```
CounterState.Subscribe(Me, "CounterChanged")

Sub CounterChanged(State As UIState)
Log("Counter = " & State.GetState)
End Sub
```

State updates are selective, not a full virtual-DOM diff. A bound widget can update itself; structural changes still require rendering the affected root.

7. Themes and customization

UITheme supplies complete light/dark defaults for colors, typography, shapes and common dimensions. A seed color can be used without configuring every widget individually.

```
Private AppTheme As UITheme
AppTheme.Initialize.Scheme(0xFF6750A4)

Dim primary As UIButton
primary.Initialize.Text("Continue").ApplyTheme(AppTheme)
```

Other initialization forms are:

```
AppTheme.InitializeDark
AppTheme.InitializeWithScheme(0xFF6750A4)
AppTheme.InitializeWithSchemeAndMode(0xFF6750A4, True)
```

Switching mode keeps application state intact:

```
AppTheme.Toggle
activeScreen.ApplyTheme(AppTheme)
```

Theme properties include Background, Surface, SurfaceVariant, PrimaryText, SecondaryText, Accent, Negative, Divider, Border, ButtonText, DashboardBar and their readable Text counterparts. Shape tokens include ButtonRadius, CardRadius, InputRadius, FabRadius and SnackbarRadius.

Explicit setters override only the property they change:

```
primary.CornerRadius(8dip).TextSize(16)
primary.ApplyTheme(AppTheme)
```

The button keeps following the theme for other properties. Containers forward ApplyTheme to their descendants.

8. Buttons, inputs and callbacks

Callbacks use normal B4A target-plus-sub-name conventions. No lambda or custom event language is required.

```
Dim saveButton As UIButton
saveButton.Initialize _
.Text("SAVE") _
.CornerRadius(12dip) _
.OnClick(Me, "Save_Click")

Sub Save_Click
Log("Saved")
End Sub
```

The callback must be a parameterless sub for UIButton and UIFloatingActionButton. Invalid callback names are ignored safely.

UIInput exposes a native B4A EditText while participating in declarative layout:

```
Private NameState As UIState
NameState.Initialize("")

Dim nameInput As UIInput
nameInput.Initialize _
.Hint("Your name") _
.BindText(NameState) _
.OnTextChanged(Me, "Name_Changed")

Sub Name_Changed(NewText As String)
NameState.SetState(NewText)
End Sub
```

The input binding is one-way. User edits arrive through OnTextChanged; the host decides whether to update state. Programmatic changes do not invoke the user-edit callback.

Rounded controls use the same fluent style:

```
nameInput.CornerRadius(10dip).Border(1dip, 0xFFD0D7E2)
saveButton.BackgroundColor(0xFF6750A4).TextColor(Colors.White)
```

CornerRadius(0) preserves the native background. Custom backgrounds can replace the native ripple; use the default background when platform ripple feedback is more important than custom shaping.

9. App structure, navigation and safe area

The recommended structure is one real B4A Activity containing virtual screens:

```
Dim appBar As UIAppBar
appBar.Initialize.Title("Home")

Dim screen As UIScaffold
screen.Initialize _
.AppBar(appBar) _
.Body(screenBody) _
.FloatingActionButtonRight(actionButton)

Navigator.Initialize _
.AddScreen("Home", screen) _
.AddScreen("Settings", settingsScreen)
Navigator.SetParent(Activity)
Navigator.SetPosition(0, 0)
Navigator.SetSize(Activity.Width, Activity.Height)
Navigator.NavigateTo("Home")
Navigator.Render
```

UINavigator manages virtual widget trees, not physical Activities or .bal files. It uses the available B4A content rectangle so the root stays below the Android status area. UIScaffold lays out the app bar, body, optional bottom navigation and FABs without making each screen calculate those offsets.

Bottom navigation is data-driven:

```
Dim tabs As UIBottomNavigationBar
tabs.Initialize _
.AddItem("home", "?", "Home") _
.AddItem("settings", "?", "Settings") _
.OnSelected(Me, "Tab_Selected")

Sub Tab_Selected(Index As Int, Id As String)
Navigator.NavigateTo(Id)
End Sub
```

10. Feedback, dialogs and animation

A snackbar is an overlay, not a permanent child of a Column:

```
Dim snack As UISnackBar
snack.Initialize _
.Message("Settings saved") _
```

```
.Action("UNDO", Me, "Undo_Click") _
.Duration(3000) _
.Show(Activity)
```

Duration(0) keeps it visible until Dismiss. AnimationDuration(0) disables its entrance and exit transition.

UIAlertDialog uses normal B4A callbacks:

```
Dim dialog As UIAlertDialog
dialog.Initialize _
.Title("Delete item?") _
.Message("This action cannot be undone.") _
.NegativeButton("Cancel", Me, "DialogCancel") _
.PositiveButton("Delete", Me, "DialogDelete") _
.ApplyTheme(AppTheme) _
.Show(Activity)
```

Dialog buttons follow the theme shape and color tokens. The dialog is an overlay and can also host a declarative content widget through Content(widget).

UIAnimation is deliberately opt-in and animates bounds of an already-mounted native view:

```
Dim animation As UIAnimation
animation.Initialize _
.TargetView(cardView) _
.MoveTo(cardView.Left, cardView.Top + 8dip) _
.Duration(220) _
.OnCompleted(Me, "AnimationDone") _
.Start
```

It does not replace the layout system or imply automatic animation of colors, opacity or arbitrary properties.

11. Lists and scrolling

Use UIScrollView for small or variable-height content:

```
Dim feed As UIColumn
feed.Initialize.Spacing(8dip).AddChild(cardOne).AddChild(cardTwo)

Dim scroll As UIScrollView
scroll.Initialize.Child(feed)
```

Use UIListView for long, fixed-height collections. It keeps a visible window and reuses item widgets:

```
Private Records As List
Private RecordsView As UIListView

RecordsView.Initialize _
.Items(Records) _
.ItemHeight(68dip) _
.CreateItem(Me, "CreateRow") _
.BindItem(Me, "BindRow")

Sub CreateRow(Index As Int) As Object
Dim label As UILabel
label.Initialize.Size(16)
Return label
End Sub

Sub BindRow(Index As Int, ItemView As Object)
Dim label As UILabel = ItemView
label.Text("" & Records.Get(Index))
End Sub
```

UIListView is fixed-height and is not a full RecyclerView replacement. Use NotifyDataSetChanged after changing the data source.

12. Native views are still possible

The library does not create a closed ecosystem. Existing B4A controls can be inserted through UINative:

```
Dim nativeButton As Button
nativeButton.Initialize("NativeButton")

Dim native As UINative
native.Initialize(nativeButton, 140dip, 48dip)

Dim row As UIRow
row.Initialize.AddChild(native).AddChild(saveButton)
```

The native view remains a normal B4A control, so its event sub and properties remain available in the host Activity. This is important for drawers, WebViews, specialized controls and platform APIs that do not need a declarative wrapper yet.

13. Why use this instead of manual AddView?

The goal is not to claim that every screen becomes shorter. The practical benefits are consistency and reuse:

- screen structure is visible as a tree instead of scattered coordinates;
- natural measurement reduces clipping and manual height calculations;
- theme tokens avoid repeating colors, text sizes and corner radii;
- state bindings keep simple display values synchronized;
- the same containers can host different widgets;
- virtual screens can live inside one real Activity;
- native B4A views remain available when a wrapper is unnecessary.

For a tiny one-label screen, traditional B4A may be equally short. The advantage becomes clearer as screens gain repeated cards, spacing, state, theming, navigation and responsive layout.

14. Troubleshooting

A button does nothing: verify the target, exact callback name, parameter signature and that the button is not covered by another native view. UIButton callbacks are parameterless; UIInput callbacks receive NewText As String.

A label is clipped: avoid arbitrary fixed heights. Let UIColumn, UIRow, UIPadding and UICard measure the child naturally. Use UIExpanded for the flexible part of a screen.

A scroll view does not scroll: ensure the child is taller than the viewport and place the scroll view inside UIExpanded when it shares a Column with fixed content.

A screen overlaps the status bar: mount the root through UINavigator and avoid independently guessing the top inset in every screen.

A theme change appears incomplete: apply the same UITheme to the root scaffold/navigator and avoid hard-coded colors in custom widgets. Explicit per-widget overrides intentionally remain unchanged.

The runtime still shows old behavior: regenerate DeclarativeUI.b4xlib, replace the installed copy, refresh the B4A Libraries Manager and compile again. The IDE does not use changed source files that are absent from the active package.

15. Minimal counter example

```
Sub Process_Globals
Private CounterState As UIState
End Sub

Sub Activity_Create(FirstTime As Boolean)
Dim root As B4XView = Activity
Dim theme As UITheme
theme.Initialize.Scheme(0xFF6750A4)

If FirstTime Then CounterState.Initialize(0)

Dim value As UILabel
```

```
value.Initialize.BindText(CounterState).Size(52).ApplyTheme(theme)

Dim plus As UIFloatingActionButton
plus.Initialize.Text("+").OnClick(Me, "Increment_Click").ApplyTheme(theme)

Dim center As UICenter
center.Initialize.Child(value)

Dim screen As UIScaffold
screen.Initialize.Body(center).FloatingActionButtonRight(plus).ApplyTheme(theme)
screen.SetParent(root)
screen.SetPosition(0, 0)
screen.SetSize(root.Width, root.Height)
screen.Render
End Sub

Sub Increment_Click
Dim currentValue As Int = CounterState.GetState
CounterState.SetState(currentValue + 1)
End Sub
```

This small example demonstrates the central idea: ordinary B4A code describes state, widgets and composition, while the library owns native mounting and layout.