

Skill Display Construction - Dashboard

Purpose

Dashboard displays are the paradigm for **data cards, not diagrams**. Rows and columns of gauges, trends, KPIs, tables. Every element lives in a `Cell`, and cells are arranged in a responsive grid that reflows on window resize.

Use Dashboard when:

- The display is primarily **data monitoring** (live values, trends, alarms)
- The layout is a **grid of cards** — fleet status, shift summary, KPI wall, operator console
- The screen must reflow across window sizes
- There's no spatial/physical relationship between equipment to preserve

Use Canvas (load **Skill Display Construction — Canvas** instead) when equipment positions relative to each other matter (pipe A connects to vessel B), you're building a P&ID, or you need animation dynamics.

Prerequisite: load **Skill Display Construction — Basics** first.

Section 1 — Dashboard mental model

A Dashboard is a grid of cells

The whole display is defined by:

1. A **DashboardDisplay** root with `Columns` and `Rows` arrays describing the grid track sizes
2. A **Cells** array, where each cell specifies `Row`, `Col`, optional `RowSpan/ColSpan`, a `Cell.HeaderLink` for the card title, and `Content` (the element that fills the cell)

```
<ac:structured-macro ac:name="code"> <ac:parameter ac:name="language">json</ac:parameter> ac:plain-text-body{ "Name": "OperationsOverview",
"PanelType": "Dashboard", "DashboardDisplay": { "Columns": [ "*" , "*" , "*" ], "Rows": [ "Auto" , "*" , "*" ], "Cells": [ { "Row": 0, "Col": 0, "ColSpan": 3, "Cell.
HeaderLink": "Plant Overview", "Content": { /* header card */ } }, { "Row": 1, "Col": 0, "Cell.HeaderLink": "Production Rate", "Content": { /* gauge */ } }, {
"Row": 1, "Col": 1, "Cell.HeaderLink": "Active Alarms", "Content": { /* alarm viewer */ } }, { "Row": 1, "Col": 2, "Cell.HeaderLink": "Shift Output", "Content": {
/* KPI */ } } ] } } }</ac:plain-text-body> </ac:structured-macro>
```

Track sizing

The `Columns` and `Rows` arrays define track sizes. Each entry can be:

- `"*"` — equal share of remaining space (like CSS `1fr`)
- `"2*"` — twice the share (like CSS `2fr`)
- `"Auto"` — size to content
- `"240"` — fixed pixel width
- `"*, min=200"` — fractional with a minimum in pixels

Standard patterns:

Layout	Columns	Rows
3 equal columns, 2 rows	["*" , "*" , "*"]	["*" , "*"]
Header + 2 equal rows	["*" , "*" , "*"]	["Auto" , "*" , "*"]
Sidebar + main	["240" , "*"]	["*"]
Header + sidebar + main	["240" , "*"]	["Auto" , "*"]
4 KPIs across top, trend below	["*" , "*" , "*" , "*"]	["Auto" , "*"]
Operator wall (4x3)	["*" , "*" , "*" , "*"]	["*" , "*" , "*"]

Don't think in pixels — think in cells

In Canvas you place a gauge at `Left: 472, Top: 320`. In Dashboard you place a gauge at `Row: 1, Col: 2`. The engine handles pixel positioning, cell padding, and reflow.

This is a completely different mental model. If you find yourself calculating `Left/Top/Width/Height` for Dashboard elements, stop — you're building a Canvas display by accident.

Cells size to their content, not the other way around

Setting a row or column to "*" gives that track the space; it does not force the cell's content to fill it. A fixed-size gauge in a "*" row will sit at its native size inside a large cell, not stretch. For content that should fill the cell, use stretch-friendly controls (TrendChart, BarChart, AlarmViewer, DataGrid). Gauges are fixed size; place them in cells sized to match.

Dashboard-compatible controls only

Not every element type works inside a Dashboard cell. Call `listElements('Dashboard')` for the authoritative compatibility list in the current release. Rule of thumb:

- **Works in Dashboard:** Interaction, Charts, Gauges, Viewer, Editors, TextBlock, Label.
- **Canvas-only:** shape primitives, first-class auto-shapes, and containers.

Dynamics work in Dashboard for visual controls (FillColorDynamic on a TextBlock, VisibilityDynamic on a chart) but the animation dynamics (Rotate, Scale, MoveDrag, Skew, Bargraph) are **Canvas-only** — they're silently ignored in Dashboard cells.

Section 2 — Standard grid recipes

3x2 KPI wall (the most common)

Six data cards in a 3-column x 2-row grid:

```
json { "Name": "PlantKPIs", "PanelType": "Dashboard", "DashboardDisplay": { "Columns": [ "*", "*", "*" ], "Rows": [ "*", "*" ], "Cells": [ { "Row": 0, "Col": 0, "Cell.HeaderLink": "Production Rate", "Content": { "Type": "CenterValue", "LinkedValue": "@Tag.Plant/ProductionRate", "CenterTextFormat": "N1", "AccentTextLink": "units/hr" } }, { "Row": 0, "Col": 1, "Cell.HeaderLink": "Active Alarms", "Content": { "Type": "CenterValue", "LinkedValue": "@Tag.Plant/AlarmCount", "CenterTextFormat": "N0", "AccentTextLink": "alarms" } }, { "Row": 0, "Col": 2, "Cell.HeaderLink": "Operator on Duty", "Content": { "Type": "TextBlock", "LinkedValue": "@Tag.Shift/CurrentOperator", "FontSize": 22, "FontWeight": "Bold" } }, { "Row": 1, "Col": 0, "Cell.HeaderLink": "Temp Trend", "Content": { "Type": "TrendChart", "Duration": "5m", "Pens": [ { "Type": "TrendPen", "LinkedValue": "@Tag.Plant/AvgTemp", "Stroke": "#FFEF4444" } ] }, { "Row": 1, "Col": 1, "Cell.HeaderLink": "Pressure Trend", "Content": { "Type": "TrendChart", "Duration": "5m", "Pens": [ { "Type": "TrendPen", "LinkedValue": "@Tag.Plant/AvgPressure", "Stroke": "#FF38BDF8" } ] }, { "Row": 1, "Col": 2, "Cell.HeaderLink": "Shift Output", "Content": { "Type": "BarChart", "LinkedValue": "@Tag.Shift/OutputByHour" } } ] }
```

4x3 operator wall

Twelve cells. Full-screen control-room overview with one card per reactor / line / zone:

```
json "Columns": [ "*", "*", "*", "*" ], "Rows": [ "*", "*", "*" ]
```

2-column detail (list detail)

Asset tree on the left, detail panel on the right:

```
json { "DashboardDisplay": { "Columns": [ "240", "*" ], "Rows": [ "Auto", "*" ], "Cells": [ { "Row": 0, "Col": 0, "ColSpan": 2, "Cell.HeaderLink": "Production Area A", "Content": { "Type": "TextBlock", "LinkedValue": "{@Tag.Site/Name} — {@Now}" } }, { "Row": 1, "Col": 0, "Cell.HeaderLink": "Equipment", "Content": { "Type": "AssetsTree" } }, { "Row": 1, "Col": 1, "Cell.HeaderLink": "@Client.Context.AssetName", "Content": { "Type": "ChildDisplay", "DisplayLink": "EquipmentDetail" } } ] }
```

The AssetsTree has all its bindings pre-wired to `@Client.Context.*` by default — drop it in and navigation "just works."

ColSpan / RowSpan for asymmetric layouts

```
json { "Row": 0, "Col": 0, "ColSpan": 2, ... } // cell spans columns 0 and 1 { "Row": 1, "Col": 0, "RowSpan": 2, ... } // cell spans rows 1 and 2
```

Use ColSpan on a header card to stretch it across all grid columns. Use RowSpan when you want a tall element (AlarmViewer, AssetsTree) next to shorter cards.

Section 3 — Controls by cell purpose

Cell purpose	Best control	Why
Display title / section header	TextBlock with composite LinkedValue	"{@Tag.Site/Name} — {@Now}" in one element
Single-value KPI (numeric)	CenterValue	Pre-styled big-number-plus-unit tile
Single-value KPI (string)	TextBlock with FontSize 24+	Simpler than CenterValue for strings
Time-series trend	TrendChart	The workhorse
Tag value dial	RadialGauge	Gauge with threshold bands
Tag value bar (linear)	LinearGauge	For bar-style KPIs with setpoint pointer

Cumulative count / meter	DigitalGauge or DigitalMeter	Specialty numeric readouts
Categorical comparison	BarChart	e.g., output by shift
Current alarms	AlarmViewer	Pre-wired to Client.AlarmPage context
Asset hierarchy navigation	AssetsTree	Pre-wired to Client.Context
Dropdown selector	ComboBox (DataTable-backed for FK)	Zero-script FK lookup
Data table / list	DataGrid	The list in listdetail
Document viewer (SOP, work order)	PdfViewer	URL-bound inline PDF
Fleet/plant map	MapsOSM	Lat/long-positioned markers
Embedded another display	ChildDisplay	Reusable detail panels
Multiple panels user switches	TabControl	Manual click-to-switch
Slideshow of status boards	Carousel	Auto-cycle with AutoCycleLink
Collapsible panel	Expander	Click to expand advanced settings
Card-per-item layout	FlowPanel	N items from a data source, one template

The Cell.HeaderLink pattern

Every cell should have a Cell.HeaderLink. The value is either:

- A static string: "Production Rate"
- A bound value: "@Tag.CurrentAsset/Name"
- A composite: "Reactor R-101: {@Tag.R101/Status}"

The header renders as a small title strip at the top of the card, in the theme's card-header style. Leaving it empty gives you a headerless card — useful for the header / status row banding the top of the display, but unusual elsewhere.

Section 4 — The TrendChart recipe

Most dashboard cells end up being trend charts. The canonical setup:

```
json { "Type": "TrendChart", "Duration": "5m", "YMinValue": 0, "YMaxValue": 100, "YLabels": 5, "XGridLines": 6, "YGridLines": 5, "LegendPlacement":
"BottomPanel", "VerticalCursor": true, "BackgroundTheme": "ControlBackground", "ForegroundTheme": "TextForeground", "BorderBrushTheme":
"DefaultBorder", "Pens": [ { "Type": "TrendPen", "LinkedValue": "@Tag.Reactor/Temperature_C", "PenLabel": "Temperature", "Stroke": "#FFEF4444",
"StrokeThickness": 2 }, { "Type": "TrendPen", "LinkedValue": "@Tag.Reactor/Setpoint", "PenLabel": "Setpoint", "Stroke": "#FF34D399", "StrokeThickness":
1 } ] }
```

Pens accepts BOTH forms

- Flat array: "Pens": [{ ... }, { ... }] prefer this for readability
- Wrapper object: "Pens": { "Type": "TrendPenList", "Children": [{ ... }] } older form, also accepted

TrendPen properties

Property	Required	Notes
LinkedValue	?	Tag binding: @Tag.Reactor/Temperature_C
PenLabel	—	Legend text
Stroke	—	Line color hex. Not theme-aware — hex only.
StrokeThickness	—	Default 1; use 2 for emphasis, 3 for setpoint/limit
YMin / YMax	—	Per-pen Y scale (overrides chart scale)
YAxisPosition	—	"Left" (default) or "Right" for dual-axis plots
Auto	—	true for auto-scaling that pen
Visible	—	true/false

Duration defaults

- "Text" — hardcoded comma-separated inline options

Section 7 — AlarmViewer (the default template is your friend)

The AlarmViewer default template ships pre-wired to `@Client.AlarmPage.*` context tags:

```
json { "Type": "AlarmViewer", "ShowRowSelectorPane": false, "BackgroundTheme": "ControlBackground", "ForegroundTheme": "TextForeground" }
```

That's the entire cell content. All alarms, all columns, all features — wired.

Custom filtering

```
json { "Type": "AlarmViewer", "Filter": "Area = 'ReactorZone'", "ShowRowSelectorPane": false }
```

Filter syntax: `"Priority >= 2", "Area = 'Tank1'"`, or a tag binding `"@Tag.AlarmFilter"`.

Custom columns

```
json "Columns": { "Type": "GridColumnList", "Children": [ { "Type": "GridColumn", "Title": "Active", "FieldName": "ActiveTime_Ticks", "Width": 140 }, {
  "Type": "GridColumn", "Title": "Tag", "FieldName": "TagName", "Width": 200 }, { "Type": "GridColumn", "Title": "Msg", "FieldName": "Message", "Width":
  400 }, { "Type": "GridColumn", "Title": "Pri", "FieldName": "Priority", "Width": 50 } ] }
```

Available field names: `AckStatus`, `ActiveTime_Ticks`, `TagName`, `Group`, `Value`, `ID`, `ItemName`, `State`, `AckRequired`, `Condition`, `SolutionName`, `Area`, `Priority`, `NormTime_Ticks`, `AckTime_Ticks`, `UserName`, `Message`, `Duration`, `Category`, `DateCreated_Ticks`, `AuxValue`, `AlarmLimit`, `PreviousValue`, `AuxValue2`, `AuxValue3`. Tick fields format as `DateTime` at render time.

Section 8 — AssetsTree + ChildDisplay navigation

The canonical asset-driven master-detail pattern. One "detail template" display shows whatever asset the operator picks in the tree.

The tree (drop it in, no config)

```
json { "Type": "AssetsTree" }
```

The default template already binds:

- `LinkedValue @Client.Context.AssetName` (output — which asset is selected)
- `AssetPathLink @Client.Context.AssetPath` (output — full path)

The detail panel — ChildDisplay

```
json { "Type": "ChildDisplay", "DisplayLink": "EquipmentDetailTemplate" }
```

Inside `EquipmentDetailTemplate`, every binding uses `Asset(@Client.Context.AssetPath + ".Property")` or direct `@Tag` paths constructed from the context. When the operator clicks a different tree node, the `ChildDisplay` re-renders with the new asset's data.

Dynamic ChildDisplay

```
json { "Type": "ChildDisplay", "DisplayLink": "@Tag.DetailDisplayName" }
```

A Script calculates which detail display to show based on the selected asset's type (`Pump PumpDetail`, `Reactor ReactorDetail`) and writes to `@Tag.DetailDisplayName`. The `ChildDisplay` swaps automatically.

Section 9 — Carousel and TabControl

Both accept the same structure: `HeaderElements` (navigation chrome) and `TabItems` (panels). Difference:

- **Carousel** — `AutoCycleLink: 5` for 5-second auto-cycling. Lobby displays, rotating KPI boards.
- **TabControl** — no auto-cycle; operator clicks tabs. Drill-down detail panels.

```
<ac:structured-macro ac:name="code"> <ac:parameter ac:name="language">json</ac:parameter> <ac:plain-text-body> { "Type": "Carousel",
  "AutoCycleLink": 5, "TabItems": [ { "Type": "TabItem", "IsSelected": true, "Header": { "Type": "TextBlock", "LinkedValue": "Production" }, "Children": [ {
    "Type": "TrendChart", ... } ] }, { "Type": "TabItem", "Header": { "Type": "TextBlock", "LinkedValue": "Quality" }, "Children": [ { "Type": "BarChart", ... } ] }, {
    "Type": "TabItem", "Header": { "Type": "TextBlock", "LinkedValue": "Alarms" }, "Children": [ { "Type": "AlarmViewer" } ] } ] } } </ac:plain-text-body> </ac:
  structured-macro>
```

Rules:

- Only ONE `TabItem` should have `IsSelected: true`
- `Children` is an array — can contain multiple elements per tab
- `AutoCycleLink` can be a number (seconds) OR a tag binding (`"@Tag.ShowroomCycleSeconds"`) for runtime-configurable cycling

Section 10 — Common KPI card recipes

Big-number CenterValue

```
json { "Type": "CenterValue", "LinkedValue": "@Tag.Plant/ProductionRate", "CenterTextFormat": "N1", "AccentTextLink": "units/hr", "CenterFontSize": 48, "AccentFontSize": 14, "BackgroundTheme": "ControlBackground" }
```

CenterTextFormat: "N0" (integer), "N1" (1 decimal), "N2" (2 decimals), "P1" (percent 1 decimal). AccentTextLink is the unit or caption under/beside the main number.

Composite TextBlock (value + unit + context in one)

```
json { "Type": "TextBlock", "LinkedValue": "Throughput: {@Tag.Plant/ProductionRate} units/hr ({@Tag.Plant/TargetPct}% of target)", "FontSize": 16, "FontWeight": "SemiBold", "ForegroundTheme": "TextForeground" }
```

Threshold-colored value

```
json { "Type": "TextBlock", "LinkedValue": "@Tag.Plant/AvgTemp" °C", "FontSize": 32, "Dynamics": [ { "Type": "TextColorDynamic", "LinkedValue": "@Tag.Plant/AvgTemp", "ChangeColorItems": [ { "Type": "ChangeColorItem", "ChangeLimit": 0, "LimitColor": "#FF38BDF8" }, { "Type": "ChangeColorItem", "ChangeLimit": 75, "LimitColor": "#FF34D399" }, { "Type": "ChangeColorItem", "ChangeLimit": 90, "LimitColor": "#FF59E0B" }, { "Type": "ChangeColorItem", "ChangeLimit": 100, "LimitColor": "#FFE444" } ] } ] }
```

Blue cold green normal amber warning red alarm.

Section 11 — Navigation on Dashboard displays

Same rule as Canvas: **page-to-page navigation belongs in the Header display**, not on content pages. Content pages get only in-page interactions.

Tab-bar header pattern

```
json { "Name": "Header", "PanelType": "Dashboard", "DashboardDisplay": { "Columns": [ "Auto", "Auto", "Auto", "Auto", "Auto", "Auto" ], "Rows": [ "*" ], "Cells": [ { "Row": 0, "Col": 0, "Content": { "Type": "Button", "LabelLink": "Overview", "Dynamics": [ { "Type": "ActionDynamic", "MouseLeftButtonDown": { "Type": "DynamicActionInfo", "ActionType": "OpenDisplay", "ObjectLink": "OperationsOverview" } } ] }, { "Row": 0, "Col": 1, "Content": { "Type": "Button", "LabelLink": "Alarms", ... } }, { "Row": 0, "Col": 2, "Content": { "Type": "Button", "LabelLink": "Trends", ... } }, { "Row": 0, "Col": 3, "Content": { "Type": "Button", "LabelLink": "Reports", ... } }, { "Row": 0, "Col": 5, "Content": { "Type": "TextBlock", "LinkedValue": "Logged in: {@Client.Username}" } } ] }
```

Minimum button size: 100×32. Recommended: 130×40 for touch-friendly control rooms.

Row-click drill-down

For DataGrids, a double-click to drill into detail:

```
json "Dynamics": [ { "Type": "ActionDynamic", "MouseDoubleClick": { "Type": "DynamicActionInfo", "ActionType": "OpenDisplay", "ObjectLink": "ReactorDetail" } } ]
```

The target display reads `@Tag.SelectedReactor.Name` (etc.) — so you get "double-click row open detail view of that row" with zero code.

Section 12 — Dashboard-specific quirks

1. **Dynamics that don't work in Dashboard.** Silently ignored: `RotateDynamic`, `ScaleDynamic`, `MoveDragDynamic`, `SkewDynamic`, `BargraphDynamic`. If you need a rotating element or a custom level-bar, move that cell's content to a Canvas display embedded via `ChildDisplay`, or pick a control with the behavior built in (`LinearGauge` for level, symbols for motor-running-spin).
2. **Cell headers use theme styles you can't directly override.** `Cell.HeaderLink` renders in a theme-defined style — you can't set `FontSize` or `Foreground` on the header itself from the cell. If you need a custom header, set `Cell.HeaderLink` to empty string and put a `TextBlock` as the first element inside the cell `Content`.
3. **ComboBox + DataTable source loads on first render.** Expect a brief empty state on display open. For critical selection flows, bind to `@Tag.SelectedValueLink` with a pre-populated default.
4. **ChildDisplay depth limit.** Don't nest `ChildDisplays` more than 2 levels deep. Beyond that, performance degrades and context-tag reuse becomes confusing.

Section 13 — Dashboard checklist

- ? `PanelType`: "Dashboard" is set
- ? `DashboardDisplay` object includes `Columns`, `Rows`, `Cells`
- ? Every cell has `Row` and `Col`; `Cell.HeaderLink` is either a non-empty string or absent
- ? No Canvas-only element types (`Rectangle`, `Ellipse`, `Polygon`, `Cylinder`, `ShapeGroup`, `SvgGroup`, `Group`) in `Content`
- ? No Canvas-only dynamics (`Rotate`, `Scale`, `MoveDrag`, `Skew`, `Bargraph`) on cell content
- ? Text values 14 `FontSize`; big KPIs 22 `FontSize`
- ? `ColSpan`/`RowSpan` summed values don't exceed the grid
- ? `AlarmViewer` uses default template (no `Columns` override) unless you NEED custom columns

- ? DataGrid SelectedValuesLink UserType-typed Client tag detail cells binding chain verified
- ? ComboBox with DataTable source has DisplayMember AND SelectedValuePath AND SelectedValueLink
- ? TrendChart Pens are flat array form; each pen has at minimum LinkedValue + Stroke
- ? Header display owns all page-to-page navigation
- ? get_state after write shows errorList empty

Section 14 — Quick reference

The 10 controls you'll actually use on most Dashboards

text TextBlock // headers, composite KPIs, status text CenterValue // big-number KPI tiles TrendChart // all time-series (the workhorse) BarChart // categorical comparison AlarmViewer // alarm list with default template AssetsTree // plant navigation sidebar ChildDisplay // embedded detail panel DataGrid // list in list-to-detail ComboBox // FK selector dropdown RadialGauge // circular gauge cells

Canonical dashboard envelope