

omnis noCode

**What Is It?**



# Overview

- An online tool to allow **anyone** to create **simple** applications.
  - *No coding necessary (or supported!)*
- Built on top of Omnis Studio.
  - *It's a JS Client app.*
- Generates an Omnis library.
  - *Can be downloaded and extended with Omnis Studio.*
  - *Generates clean, commented code.*

# UX Requirements



# Fundamental Questions

Without any code:

- How can the user **refer to data**?
- How can the user specify **application logic**?
- How can they build **conditional logic**?

These areas need intuitive UI.

- Our design partners at *PixelStress* helped us come up with a solution.

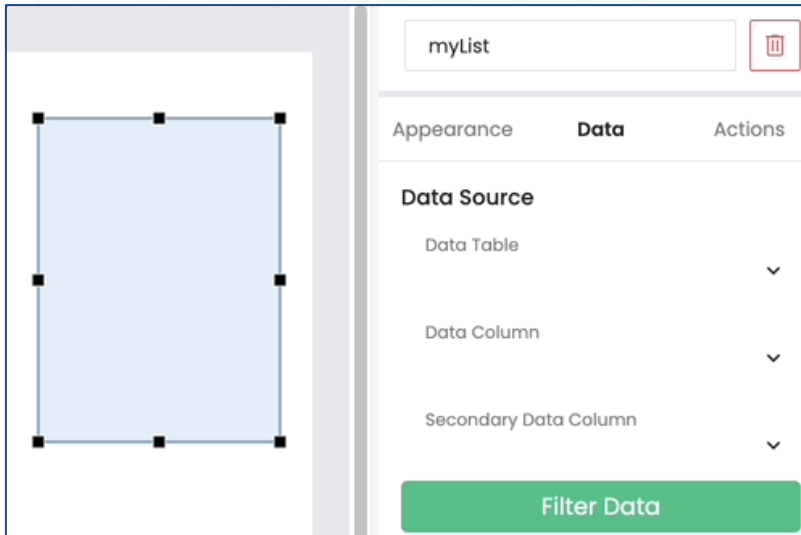
# Defining Data

Data can be defined as **Tables** in the **Data view**.

- Either manually or imported.

**Variables** can also be defined.

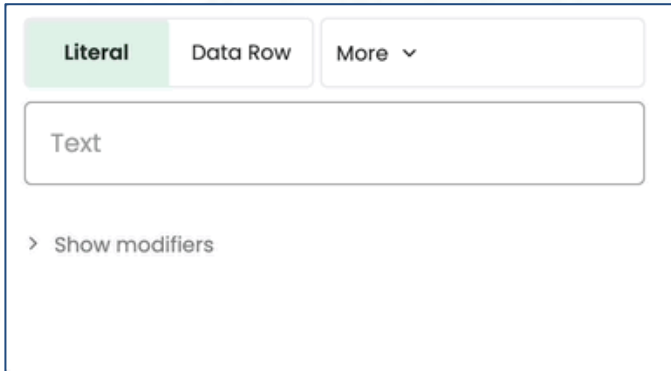
- Either 'Form' scoped or 'Global'.



- A preview of the data is shown at design time.

# Referencing Data

## General Purpose 'Data Source' View



The image shows a user interface for a 'Data Source' view. It features a horizontal tab bar with three tabs: 'Literal' (highlighted in green), 'Data Row', and 'More' (with a dropdown arrow). Below the tabs is a large text input field containing the word 'Text'. At the bottom left of the interface is a link that says '> Show modifiers'.

- Same view used everywhere the user can reference data.
- Different Data Source 'types' are shown, dependant on **context**.
- The user should become familiar with this view, wherever they see it.

# Referencing Data

## Lists and Rows

We tried to keep things simple for the user:

- Generally, a list-based control creates an implicit **List variable**, named as the table.
- Also creates a **Row variable** corresponding to the *current row* of the list.

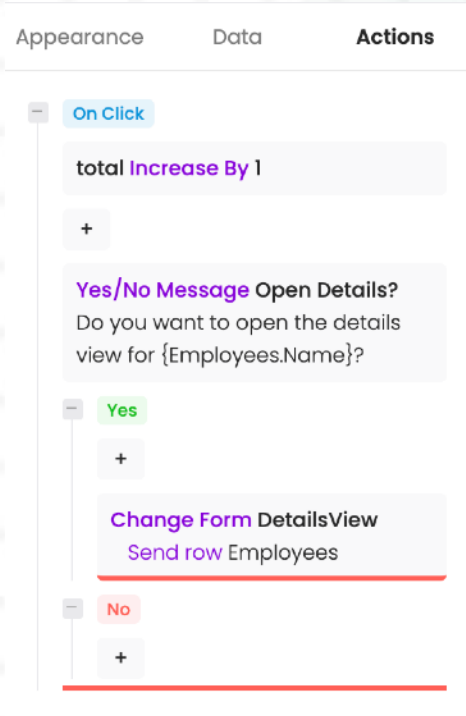
Many situations can be handled only using these.

- **Variables** are available for more fine-grain control.



# Building App Logic

## Graphical 'Actions View':



- A custom 'JSON-Defined Control'.
  - Uses a 'Tree' data structure - available on Github: [github.com/OmnisStudio/Omnis-Tree-Data-Structure](https://github.com/OmnisStudio/Omnis-Tree-Data-Structure)
- Allows user to add/edit 'actions'.
  - Action: customizable predefined actions.
- Displays the flow of logic as a list of simple card-like views of actions.
  - Supports **branching** logic.
    - Collapsible sections.
  - "+" icons show the point new actions can be inserted.
  - Drag & drop to re-order.

# Condition Editor

Needed a **simple** way to visualize & edit compound conditional statements.

- A condition can be broken down to:

Left Side	Operator	Right Side
-----------	----------	------------

- The user needs to be able to **group** multiple conditions:

*If (A AND B) OR C*

# Condition Editor

Table Column  
If Employees Name starts with "a" ← Condition 1

OR

Table Column Variable, Integer  
Employees Age is greater than total ← Condition 2

AND OR

Group 1 - View Mode

Data Row Variable More ▾

Data Row Data Column Operator Variable

Employees Employees Name is equal to defaultName (Global) +

▼ Show modifiers

Casing Normal ▼ Casing Ignore Case ▼

Add condition

Add group

Group 2 - Edit Mode



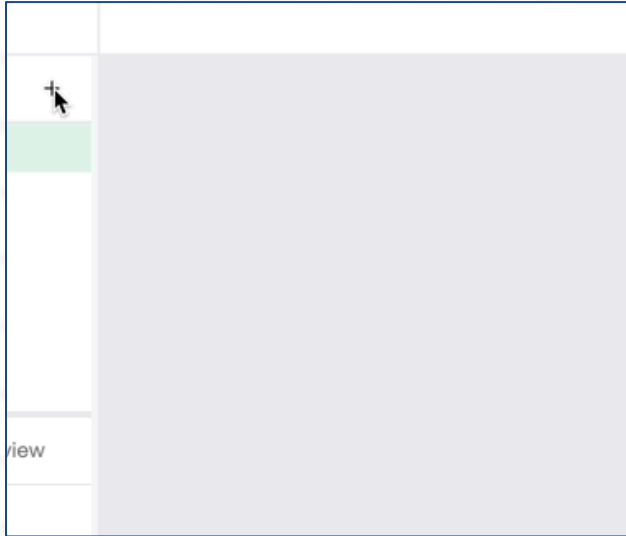
**Benefits To Studio**

# Subform Set Palettes

New “*subformpaletteshow*” \$clientcommand.

- Opens a subform dialog positioned next to a given control.
- With an arrow pointing at that control.

Useful to maintain context for an opened Subform dialog.



# Subform Loading Promises

No Code makes *heavy* use of Subforms.

- In many cases we want to load a subform, then send it a message.
  - *With many subforms, this was more complex than is ideal.*

Now, if you assign a Subform's `$classname` **on the client**, it will return a **Promise**.

- The promise will **resolve** once the subform instance has loaded.  
(Requires a *little* JavaScript)

```
Do $cinst.$objs.SubForm.$classname.$assign("jsSubFormDialog") Returns IPromise
JavaScript:IPromise.then( () => {
Do $cinst.$objs.SubForm.$subinst().$myMethod()
JavaScript:})
```

# Dialog Promises

The promise support has been extended to dialogs.

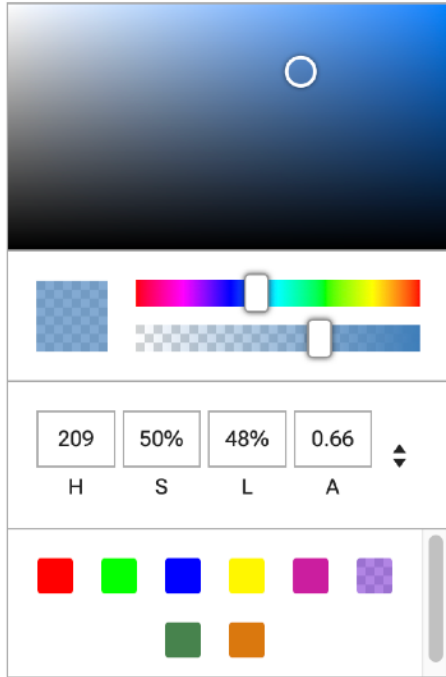
- `$showmessage()`
- `yesnomessage`, `noyesmessage` & `javamessage` `$clientcommands`.

When called **on the client**, they return a Promise.

- Resolved when the dialog is closed.
- A return value (true/false) is passed for `yesno`/`noyesmessage`.

```
Do $cinst.$clientcommand("yesnomessage",row("Are you sure?")) Returns IPromise
JavaScript:IPromise.then( (result) => {
JavaScript:IResult = result; // Store the JS 'result' parameter in the Omnis local var 'IResult'
Do $cinst.$showmessage(con("You picked ",IResult))
JavaScript:});
```

# Color Picker Component

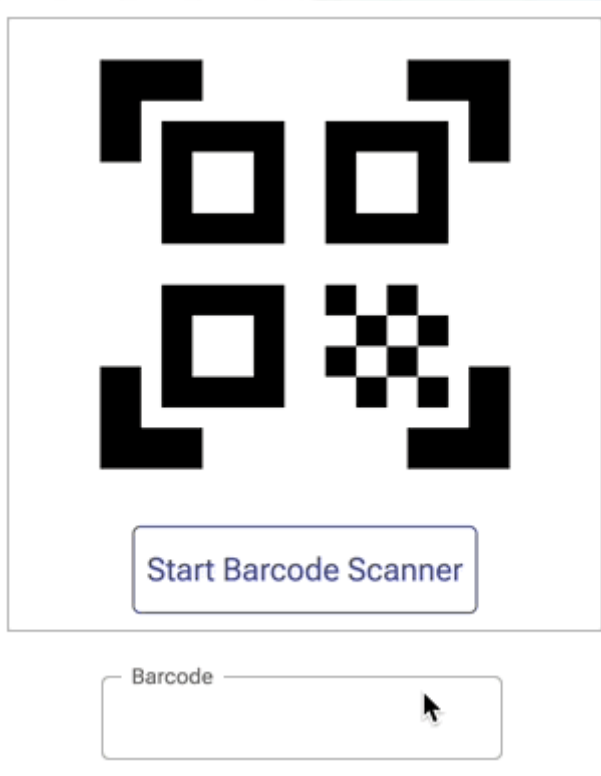


The JS Client **Color Picker** component developed due to No Code requirement.

- Allows end-user to pick a color.
  - Express in **RGB**, **HSL** or **HEX** format.
  - Optional **alpha** component.
  - Option to include 'swatch' of predefined colors.








# Camera Component

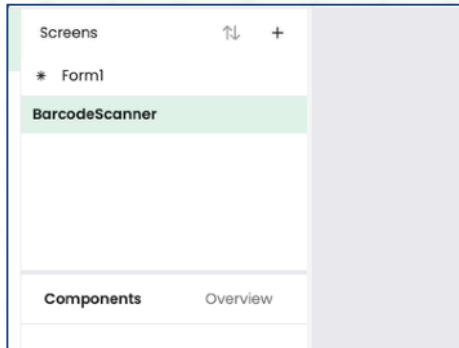


- New JS Client control to take **photos** & **scan barcodes** using native web APIs.
  - *No need for wrapper app.*
  - *Supports multiple barcode/QR code formats.*
  - *Option for default UI controls.*

# Native List Enhancements

Main Types	
 Button <small>kJSNativeListAccessoryTypeButton</small> <span>My Button</span>	
 Disclosure <small>kJSNativeListAccessoryTypeDisclosure</small> >	
 Checkbox <small>kJSNativeListAccessoryTypeCheckbox</small> ✓	
Other Types	
 Custom <small>kJSNativeListAccessoryTypeCustom</small> →	
 Custom + event <small>kJSNativeListAccessoryTypeCustomWithEvent</small> →	

- **\$reordermode** property - allows drag to re-order rows.
  - *None, Left, Right.*
  - **\$reorderbetweengroups** to control if rows can be moved between groups.



- New 'Accessory Type': **Menu**.
  - Use **\$menulistname** to statically populate for all rows.
  - **\$populatemenue** method to dynamically populate.

# JS Worker ES Module Support

JavaScript supports 2 module formats: **CommonJS** and **ES Modules**.

- JS Worker only supported CommonJS modules.
  - *Awkward to use ES Modules in CommonJS modules.*

Now **also** supports ES Modules.

- More modern format.
- Easily import CommonJS or ES Modules.
- Support top-level async/await.

Recommended format for new JS Worker modules.

- No longer need to worry about the format of other imported modules.

# New Edit Field \$labelposition

New JS Edit Field \$labelposition option:  
**kJSLabelPosInside**

- Shows the field's *\$label* text inside the bounds of the control.



A screenshot of a text input field. The field is rectangular with a thin border and rounded corners. Inside the field, the word "Name" is written in a light gray font. A mouse cursor is positioned at the end of the text, pointing towards the right.

# Server Architecture



# Containers

The No Code dev environment runs the headless server in a Docker container.

- Gives total control over runtime environment.
- Sandboxes the application.
- Each user gets their own container instance.

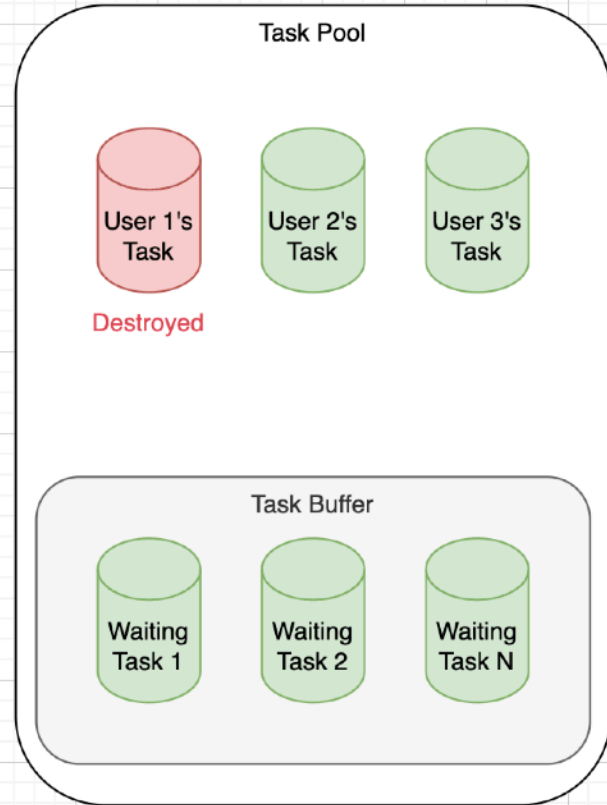
The containers are run in AWS' **Elastic Container Service** (ECS).

- ECS terms running container instances '**Tasks**'.
- We have APIs to dynamically start/stop/message Tasks.

# Task Pool

User 1 logs off or idle for too long

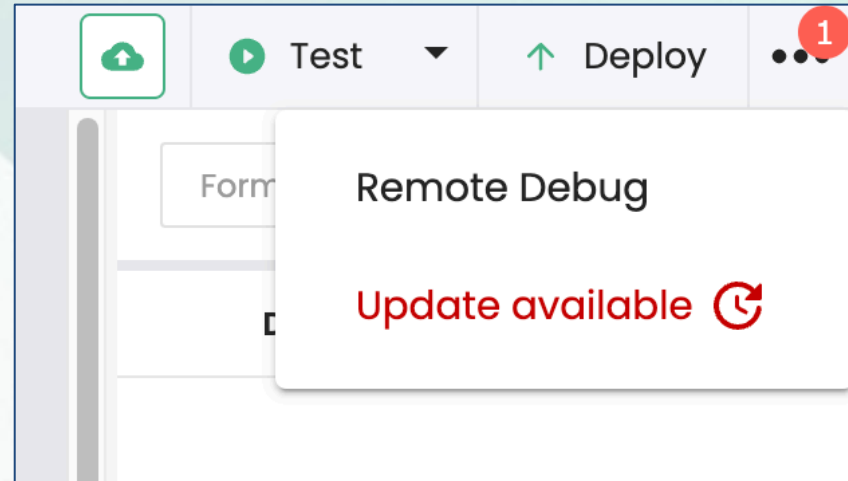
Relay Server  
[nocode.omnis.net](https://nocode.omnis.net)



# Pushing Updates

The container-based architecture allows us a seamless update mechanism.

- Updates made, rebuild Docker container.
- Push container to AWS' container 'registry'.
- Notify the Relay Server:
  - **Buffered Tasks** killed & restarted using new container.
  - **Assigned tasks** sent a 'RESTful' request - Omnis lib shows a badge to the user allowing them to restart to apply the update.





# Persisting Data

Docker containers are **ephemeral** - when they're destroyed, so is any data written to disk.

- Application information is stored in a central **Postgres DB**.
- Resource files (library, SQLite DB etc) are stored in **Amazon S3**.
  - *More efficient and cheaper than in Postgres.*
- No Code dev server dynamically downloads & uploads from S3 as required.
  - *Using JS Worker*

**Thank You**