Menu Driver - Overview Page

Introduction

The **List Page** is a navigation-oriented page type within the menu driver. It provides a structured interface for selecting between multiple entries, typically representing:

- subpages
- actions
- system modules

It is the primary mechanism for **user-driven navigation** within the menu system.

Purpose

The list page exists to answer:

> *“Where do you want to go next?”*

It is not responsible for displaying system state in detail. Instead, it:

- presents a bounded set of selectable entries
- tracks the current selection
- provides visual feedback for navigation
- enables transitions to other pages

In practice, it functions as the **entry point and routing layer** of the UI.

Architectural Role

The list page sits at the intersection of:

- **input handling** (user navigation)
- **menu structure** (page hierarchy)
- **visual rendering** (icons and labels)

```
[ Input ] → [ List Page ] → [ Page Transition ]
```

It does not consume system data (like overview pages), but rather controls **flow through the interface**.

Data Model

The list page is backed by the following state structure:

```
typedef struct {
  uint8_t num_entries;
  uint8_t selected_index;
  uint8_t entry_ids[MAX_LIST_ENTRIES];
  const uint8_t (*entry_icons)[MENU_DRIVER_ICON_BYTE_SIZE];
  bool first_render;
} page_list_state;
```

Entry Management

`num_entries`

Defines how many entries are currently active.

This value must not exceed `MAX_LIST_ENTRIES`.

`entry_ids`

Maps each visible entry to a logical identifier.

These IDs are typically used to:

- determine which page to switch to
- associate actions with selections

`entry_icons`

Pointer to icon data associated with each entry.

- Icons are rendered alongside entries
- Each icon is a fixed-size bitmap
- Icons are stored in flash as static data

Selection State

`selected_index`

Indicates which entry is currently selected.

This is the central piece of state for navigation.

All rendering and transitions depend on this value.

Render Control

`first_render`

Indicates whether the page is being rendered for the first time.

Used to:

- trigger full initial draw
- avoid redundant rendering of static UI elements

Rendering Model

The list page uses a **focused rendering strategy**, rather than displaying all entries simultaneously.

Visible Entries

Only three entries are rendered at any time:

- previous entry
- current (selected) entry
- next entry

This creates a scrolling effect without requiring full list rendering.

Rendering Optimization

The only expensive draw of the list page is the initial one which draws the selection border, all initial entries (both icons and names).

After that the only thing that gets redrawn are the icons and the texts. There is also heavier optimization done for minimal font redrawing by keeping track of previously rendered text widths.

> This is critical for SPI-driven displays, where bandwidth is limited.

Interaction Model

The list page assumes an abstract input interface:

```
menu_input (*get_input)(void);
```

The page does not interpret physical inputs directly. Instead, it operates on abstract input values, allowing it to remain independent of hardware specifics.

Expected interactions include:

- move selection up
- move selection down
- confirm selection

Relationship to Menu System

The list page enables hierarchical navigation through:

- `entry_ids` → target page identifiers
- `parent_id` (in `menu_page_t`) → upward navigation

> This allows the menu system to behave as a **tree of pages**, rather than a flat structure.

Performance Considerations

The list page is designed for constrained environments:

- partial rendering minimizes SPI usage
- static memory avoids allocation overhead
- limited visible entries reduce draw complexity

Revision #1
Created 2026-04-15 14:59:11 UTC by Nikolaos Diamantopoulos
Updated 2026-04-15 15:17:51 UTC by Nikolaos Diamantopoulos