Menu Driver - Core Data Structures

Page State Types

A *page state* is:

> The **persistent data container** that represents everything a UI page needs to function between frames.

Not just data. It's:

- memory of what the user did
- memory of what was rendered
- memory of external data (diagnostics, etc.)

List Page State

```
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;
```

Responsibilities:

- track selection
- map entries → page IDs
- hold icons
- manage first render optimization

Overview Page State

> Todo :D

State Union

```
typedef union {
  page_list_state list;
  page_overview_state overview;
} menu_page_state;
```

Page Type

```
typedef enum {
  MENU_PAGE_TYPE_LIST,
  MENU_PAGE_TYPE_OVERVIEW,
} menu_page_type_t;
```

Used to interpret the union correctly.

Page Object

```
typedef struct {
  menu_page_state *state;
  menu_page_type_t type;
  unsigned char id;
  unsigned char parent_id;
  bool needs_render;

  char name[MAX_PAGE_NAME_LEN];

  void (*init)(menu_page_state *);
  void (*update)(menu_manager_t *);
  void (*render)(menu_manager_t *);
  void (*destruct)(menu_page_state *);
} menu_page_t;
```

> This is the **core abstraction**.

Definition and Role

A page object represents one logical screen within the menu system. It encapsulates:

- the data required to represent the page (via its state)
- the functions required to manage its lifecycle
- metadata used for navigation and identification

This abstraction allows the menu system to treat all pages uniformly, regardless of their internal implementation or purpose.

Important Fields

Render Control

```
bool needs_render;
```

This flag indicates whether the page requires re-rendering.

It allows the system to avoid unnecessary redraw operations, which is critical in environments where display updates are expensive.

The responsibility for managing this flag lies with the **page implementation**.

Lifecycle Function Pointers

Each page defines its own behavior through four function pointers:

Initialization

```
void (*init)(menu_page_state *state);
```

Responsible for preparing the page state when the page becomes active.

Typical responsibilities include:

- resetting selection indices
- initializing flags
- preparing any required data structures

Update

```
void (*update)(struct menu_manager_t *manager);
```

Handles input processing and state updates.

This function is expected to:

- read input through the manager
- modify internal state accordingly
- trigger page transitions if necessary

Render

```
void (*render)(struct menu_manager_t *manager);
```

Responsible for drawing the page to the display.

This function should:

- read from the page state
- issue drawing commands via the display driver
- respect the `needs_render` flag when applicable

Destruction

```
void (*destruct)(menu_page_state *state);
```

Handles cleanup when the page is no longer active.

In embedded systems, this typically involves:

- resetting state fields
- releasing logical ownership of resources

Dynamic memory cleanup is generally not required unless explicitly used.

Menu Manager

```
typedef struct {
  unsigned char active_page_id;
  const menu_page_t *pages;
  menu_input (*get_input)(void);
} menu_manager_t;
```

Responsibilities:

- track active page

- provide input access
- hold page table

Does NOT:

- validate anything
- own memory
- manage concurrency

Revision #2
Created 2026-04-15 14:42:59 UTC by Nikolaos Diamantopoulos
Updated 2026-04-15 14:57:29 UTC by Nikolaos Diamantopoulos