# Hospital Scraper Pattern Reference Guide

## Pattern 1: h2\_name\_h3\_title (Sequential Elements)

\*\*Description:\*\* Names and titles are in separate, sequential HTML elements. Name element is followed by title element.

\*\*YAML Structure:\*\*

```yaml

pattern: "h2\_name\_h3\_title"

html\_structure:

name\_element: "h2" # ← CUSTOMIZABLE

title\_element: "h3" # ← CUSTOMIZABLE

```

\*\*Customizable Components:\*\*

- `name\_element`: Any heading tag (h1, h2, h3, h4, h5, h6), p, span, div, strong

- `title\_element`: Any heading tag (h1, h2, h3, h4, h5, h6), p, span, div, strong

\*\*Examples:\*\*

```yaml

# H3 names, H4 titles

name\_element: "h3"

title\_element: "h4"

# H2 names, P titles

name\_element: "h2"

title\_element: "p"

# Strong names, span titles

name\_element: "strong"

title\_element: "span"

```

\*\*When to Use:\*\* Names and titles are in different element types, appearing sequentially on the page.

---

## Pattern 2: combined\_h2 (Combined Name+Title)

\*\*Description:\*\* Name and title are in the SAME element, separated by a character/string.

\*\*YAML Structure:\*\*

```yaml

pattern: "combined\_h2"

html\_structure:

combined\_element: "h2" # ← CUSTOMIZABLE

separator: " - " # ← CUSTOMIZABLE

```

\*\*Customizable Components:\*\*

- `combined\_element`: Any HTML element (h1, h2, h3, h4, p, div, span, li, td)

- `separator`: Any string that separates name from title

\*\*Examples:\*\*

```yaml

# H3 with dash separator

combined\_element: "h3"

separator: " - "

# P with comma separator

combined\_element: "p"

separator: ", "

# List items with pipe separator

combined\_element: "li"

separator: " | "

# Div with colon

combined\_element: "div"

separator: ": "

```

\*\*When to Use:\*\* Name and title appear together in one element like "John Smith - CEO" or "Jane Doe, President"

---

## Pattern 3: table\_rows (Table Structure)

\*\*Description:\*\* Names and titles are in table cells, typically in different columns.

\*\*YAML Structure:\*\*

```yaml

pattern: "table\_rows"

html\_structure:

structure\_type: "table"

name\_location: "td\_column\_1" # ← CUSTOMIZABLE (column number)

title\_location: "td\_column\_2" # ← CUSTOMIZABLE (column number)

```

\*\*Customizable Components:\*\*

- `name\_location`: "td\_column\_X" or "th\_column\_X" where X is the column number (1, 2, 3, etc.)

- `title\_location`: "td\_column\_X" or "th\_column\_X" where X is the column number (1, 2, 3, etc.)

\*\*Examples:\*\*

```yaml

# Name in column 1, title in column 2

name\_location: "td\_column\_1"

title\_location: "td\_column\_2"

# Name in column 2, title in column 3

name\_location: "td\_column\_2"

title\_location: "td\_column\_3"

# Names in table headers

name\_location: "th\_column\_1"

title\_location: "td\_column\_2"

```

\*\*When to Use:\*\* Executives are listed in an HTML table with clear column structure.

---

## Pattern 4: h2\_name\_p\_title (Specific H2→P Pattern)

\*\*Description:\*\* Name in H2 element, title in the immediately following P element. More strict than Pattern 1.

\*\*YAML Structure:\*\*

```yaml

pattern: "h2\_name\_p\_title"

html\_structure:

name\_element: "h2" # Fixed as h2

title\_element: "p" # Fixed as p

```

\*\*Customizable Components:\*\*

- Not customizable - this is a specific pattern for H2→P structure

- If you need different elements, use Pattern 1 instead

\*\*When to Use:\*\* Specifically when you have `<h2>Name</h2>` followed by `<p>Title</p>` structure.

---

## Pattern 5: div\_classes (CSS Class-Based)

\*\*Description:\*\* Names and titles are in elements with specific CSS classes.

\*\*YAML Structure:\*\*

```yaml

pattern: "div\_classes"

html\_structure:

name\_class: "staff-name" # ← CUSTOMIZABLE

title\_class: "staff-title" # ← CUSTOMIZABLE

container\_class: "staff-member" # ← OPTIONAL

```

\*\*Customizable Components:\*\*

- `name\_class`: CSS class name for name elements (without the dot)

- `title\_class`: CSS class name for title elements (without the dot)

- `container\_class`: (Optional) Parent container class

\*\*Examples:\*\*

```yaml

# Standard div classes

name\_class: "executive-name"

title\_class: "executive-title"

# Span classes

name\_class: "bio-name"

title\_class: "bio-position"

# Card-based layout

name\_class: "card-title"

title\_class: "card-subtitle"

container\_class: "team-card"

```

\*\*When to Use:\*\* HTML uses semantic CSS classes like class="name" and class="title".

---

## Pattern 6: list\_items (List-Based)

\*\*Description:\*\* Names and titles are in list items (ul/ol), combined with separator.

\*\*YAML Structure:\*\*

```yaml

pattern: "list\_items"

html\_structure:

list\_type: "ul" # ← CUSTOMIZABLE (ul or ol)

format: "combined" # ← CUSTOMIZABLE (combined or sequential)

separator: " - " # ← CUSTOMIZABLE (if combined)

```

\*\*Customizable Components:\*\*

- `list\_type`: "ul" (unordered) or "ol" (ordered)

- `format`: "combined" (name-title in same li) or "sequential" (separate li elements)

- `separator`: If combined format, the separator string (supports regex for flexible whitespace)

\*\*Examples:\*\*

```yaml

# Combined list items with dash

list\_type: "ul"

format: "combined"

separator: " - "

# Pipe separator with flexible spacing

list\_type: "ul"

format: "combined"

separator: " | " # Automatically handles variations in whitespace

# Sequential list items

list\_type: "ul"

format: "sequential"

```

\*\*When to Use:\*\* Executives are in `<ul>` or `<ol>` lists with name and title combined using a separator.

\*\*Note:\*\* Pattern handles inconsistent whitespace (spaces, non-breaking spaces) automatically.

---

## Pattern 7: boardcard\_gallery (Special Gallery Pattern)

\*\*Description:\*\* Executives in card/gallery layout with specific div class, name and title separated by comma.

\*\*YAML Structure:\*\*

```yaml

pattern: "boardcard\_gallery"

html\_structure:

container\_class: "boardcard" # ← CUSTOMIZABLE

text\_format: "name\_comma\_title"

separator: "," # ← CUSTOMIZABLE

```

\*\*Customizable Components:\*\*

- `container\_class`: CSS class of the card/gallery container

- `separator`: Character separating name from title (usually comma)

\*\*Examples:\*\*

```yaml

# Standard boardcard

container\_class: "boardcard"

separator: ","

# Different gallery class

container\_class: "executive-card"

separator: " | "

# Team member cards

container\_class: "team-member"

separator: " - "

```

\*\*When to Use:\*\* Gallery or card-based layouts with combined name-title text.

---

## Pattern 8: custom\_table\_nested (Complex Nested Tables)

\*\*Description:\*\* Table structure with nested elements inside cells (like p inside td, div inside td).

\*\*YAML Structure:\*\*

```yaml

pattern: "custom\_table\_nested"

html\_structure:

structure\_type: "table\_with\_nested\_elements"

name\_selector: "td p[style\*='text-align: left']" # ← CUSTOMIZABLE

title\_selector: "td div[style\*='text-align: left']" # ← CUSTOMIZABLE

container: "td" # ← CUSTOMIZABLE

```

\*\*Customizable Components:\*\*

- `name\_selector`: Full CSS selector for name element (can include attributes, styles)

- `title\_selector`: Full CSS selector for title element

- `container`: Parent element containing both name and title

\*\*Examples:\*\*

```yaml

# Names in p, titles in div with style attribute

name\_selector: "td p[style\*='text-align: left']"

title\_selector: "td div[style\*='text-align: left']"

# Names in span with class, titles in div

name\_selector: "td span.executive-name"

title\_selector: "td div.executive-title"

# Different container

name\_selector: "div.profile p.name"

title\_selector: "div.profile span.title"

container: "div.profile"

```

\*\*When to Use:\*\* Complex table structures with nested HTML elements and specific styling.

---

## Pattern 9: field\_content\_sequential (Sequential Same-Class Elements) ⭐ NEW

\*\*Description:\*\* All data in the same CSS class, appearing in a predictable sequential pattern with a fixed step interval.

\*\*YAML Structure:\*\*

```yaml

pattern: "field\_content\_sequential"

html\_structure:

element\_selector: ".field-content" # ← CUSTOMIZABLE

pattern\_type: "sequential\_every\_3" # ← CUSTOMIZABLE

start\_index: 3 # ← CUSTOMIZABLE

```

\*\*Customizable Components:\*\*

- `element\_selector`: CSS selector for the repeating elements

- `pattern\_type`: Description of the pattern (for documentation)

- `start\_index`: Which element to start from (1-based indexing)

\*\*Pattern Logic:\*\*

- Starts at `start\_index`

- Extracts name at position `i`

- Extracts title at position `i+1`

- Skips to next name at position `i+3`

- Repeats until end of elements

\*\*Examples:\*\*

```yaml

# Every 3 elements starting at position 3

element\_selector: ".field-content"

pattern\_type: "sequential\_every\_3"

start\_index: 3

# Every 2 elements starting at position 1

element\_selector: ".bio-item"

pattern\_type: "sequential\_every\_2"

start\_index: 1

# Different class with offset

element\_selector: ".team-data"

pattern\_type: "sequential\_every\_4"

start\_index: 5

```

\*\*When to Use:\*\* Website dumps all data into the same CSS class in a predictable repeating pattern (Name, Title, Empty, Name, Title, Empty...).

\*\*Real Example:\*\* Holland Bloorview (FAC 939) - all executive data in `.field-content` elements with pattern: skip first 2, then Name, Title, Empty, repeat.

---

## Pattern 10: nested\_list\_with\_ids (ID-Based Sequential Pairing) ⭐ NEW

\*\*Description:\*\* Names and titles in separate elements identified by ID patterns or specific selectors, paired sequentially.

\*\*YAML Structure:\*\*

```yaml

pattern: "nested\_list\_with\_ids"

html\_structure:

name\_selector: "div[id^='t-']" # ← CUSTOMIZABLE

title\_selector: "span[id^='d-']" # ← CUSTOMIZABLE

container: "li.column" # ← OPTIONAL

```

\*\*Customizable Components:\*\*

- `name\_selector`: CSS selector for name elements (can use ID patterns, classes, attributes)

- `title\_selector`: CSS selector for title elements

- `container`: (Optional) Parent container element for documentation

\*\*Pairing Logic:\*\*

- Extracts ALL elements matching `name\_selector`

- Extracts ALL elements matching `title\_selector`

- Pairs them sequentially: name[1] with title[1], name[2] with title[2], etc.

\*\*Examples:\*\*

```yaml

# ID-based selectors (Baycrest style)

name\_selector: "div[id^='t-']"

title\_selector: "span[id^='d-']"

# Class-based selectors

name\_selector: "h3.executive-name"

title\_selector: "p.executive-title"

# Attribute-based selectors

name\_selector: "div[data-type='name']"

title\_selector: "div[data-type='title']"

# Complex CSS selectors

name\_selector: "div.bio-card > h4"

title\_selector: "div.bio-card > p.position"

```

\*\*When to Use:\*\*

- Names and titles are in separate, distinctly identifiable elements

- Elements appear in the same sequential order

- Need more powerful CSS selectors than simple class matching

\*\*Real Example:\*\* Baycrest (FAC 827) - names in `<div id="t-0">`, `<div id="t-1">`, etc.; titles in `<span id="d-0">`, `<span id="d-1">`, etc.

---

## Pattern Selection Decision Tree

```

Are name and title in same element?

├─ YES → Use "combined\_h2" (Pattern 2)

└─ NO → Continue...

Is it a table structure?

├─ YES →

│ ├─ Simple columns? → Use "table\_rows" (Pattern 3)

│ └─ Nested elements in cells? → Use "custom\_table\_nested" (Pattern 8)

└─ NO → Continue...

Are they in list items (ul/ol)?

├─ YES →

│ ├─ Combined with separator? → Use "list\_items" (Pattern 6)

│ └─ Nested in list structure? → Continue...

└─ NO → Continue...

Do elements have specific CSS classes?

├─ YES →

│ ├─ Different classes for name/title? → Use "div\_classes" (Pattern 5)

│ └─ Same class, sequential pattern? → Use "field\_content\_sequential" (Pattern 9) ⭐

└─ NO → Continue...

Do elements have ID patterns or need complex selectors?

├─ YES → Use "nested\_list\_with\_ids" (Pattern 10) ⭐

└─ NO → Continue...

Is it a gallery/card layout?

├─ YES → Use "boardcard\_gallery" (Pattern 7)

└─ NO → Continue...

Is it specifically H2→P structure?

├─ YES → Use "h2\_name\_p\_title" (Pattern 4)

└─ NO → Use "h2\_name\_h3\_title" (Pattern 1) with custom elements

```

---

## Pro Tips

1. \*\*Most flexible patterns:\*\* Pattern 1 (h2\_name\_h3\_title) and Pattern 2 (combined\_h2) - can adapt to many HTML structures

2. \*\*When in doubt:\*\* Start with Pattern 1 and customize the element types

3. \*\*Complex structures:\*\*

- Use Pattern 8 (custom\_table\_nested) for nested tables with CSS selectors

- Use Pattern 10 (nested\_list\_with\_ids) for elements with ID patterns or complex selectors

4. \*\*Same-class repeating data:\*\* Use Pattern 9 (field\_content\_sequential) when all data shares one class

5. \*\*Inconsistent spacing:\*\* Pattern 6 (list\_items) automatically handles variations in whitespace

6. \*\*Missing people:\*\* Works with ALL patterns - just add `missing\_people` section to YAML

7. \*\*Test before committing:\*\* Always use `helper$test\_hospital\_config()` or `quick\_test()` before adding to main YAML

---

## Pattern Summary Table

| Pattern | Best For | Complexity | Customizability |

|---------|----------|------------|-----------------|

| 1. h2\_name\_h3\_title | Sequential different elements | Low | High |

| 2. combined\_h2 | Name+title in same element | Low | High |

| 3. table\_rows | Simple table structure | Low | Medium |

| 4. h2\_name\_p\_title | Specific H2→P structure | Low | Low |

| 5. div\_classes | CSS class-based | Medium | High |

| 6. list\_items | List with separators | Medium | Medium |

| 7. boardcard\_gallery | Card/gallery layouts | Medium | Medium |

| 8. custom\_table\_nested | Complex nested tables | High | Very High |

| 9. field\_content\_sequential ⭐ | Repeating same-class pattern | Medium | Medium |

| 10. nested\_list\_with\_ids ⭐ | ID patterns, complex selectors | Medium | Very High |

---

## Debugging Tips

1. \*\*Use the diagnostic scripts\*\* to understand HTML structure before choosing a pattern

2. \*\*Check for whitespace issues\*\* - Pattern 6 now handles this automatically

3. \*\*Look for ID patterns\*\* - Pattern 10 works great with IDs like `id="t-1"`, `id="t-2"`

4. \*\*Same class everywhere?\*\* - Pattern 9 might be your answer

5. \*\*Enable debug output\*\* - Most patterns print rejection reasons to help troubleshoot