Qwikbrew Configuration Spec

Version: Q3.2.0-D1

An **in-depth reference** for authoring configuration files for **Qwikbrew**, walking you through every section, field, pattern rule, and real-world tip. Use the detailed examples and "gotchas" to ensure your config files behave exactly as you expect.

Multi-Format Support

Qwikbrew now supports multiple configuration formats:

- **YAML** (default) - `.yaml`, `.yml`
- **JSON** - `.json`
- **TOML** - `.toml`
- **XML** - `.xml`

Format Detection

Qwikbrew automatically detects your configuration format based on the file extension:

```
# Auto-detected formats
qwikbrew extract -c config.yaml    # YAML
qwikbrew extract -c config.json    # JSON
qwikbrew extract -c config.toml    # TOML
qwikbrew extract -c config.xml     # XML
```

Manual Format Override

Use the `--override-type-detection` flag to force a specific format:

```
# Force YAML parsing even with different extension
qwikbrew extract -c myconfig.txt --override-type-detection=yaml

# Force JSON parsing
qwikbrew extract -c settings.cfg --override-type-detection=json

# Force TOML parsing (case insensitive)
qwikbrew extract -c build.conf --override-type-detection=TOML

# Force XML parsing
qwikbrew extract -c settings.cfg --override-type-detection=xml
```

Supported override values: `yaml`, `yml`, `json`, `toml`, `xml` (case insensitive)

Configuration Format Examples

The same configuration can be written in any supported format:

YAML Example

```yaml
output_structure:
  "#/images":
    description: "Artwork & screens"
    create: true

zip_sources:
  - url: "https://example.com/archive.zip"
    name: "Example Archive"
    target_dir: "#/data"
```

JSON Example

```json
{
  "output_structure": {
    "#/images": {
      "description": "Artwork & screens",
      "create": true
    }
  },
  "zip_sources": [
    {
      "url": "https://example.com/archive.zip",
      "name": "Example Archive",
      "target_dir": "#/data"
    }
  ]
}
```

TOML Example

```toml
[output_structure."#/images"]
description = "Artwork & screens"
create = true

[[zip_sources]]
url = "https://example.com/archive.zip"
name = "Example Archive"
target_dir = "#/data"
```

XML Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <output_structure>
    <dir path="#/images">
      <description>Artwork &amp;amp; screens</description>
      <create>true</create>
    </dir>
  </output_structure>
  <zip_sources>
    <source>
```

```
      <url>https://example.com/archive.zip</url>
      <name>Example Archive</name>
      <target_dir>#/data</target_dir>
    </source>
  </zip_sources>
</root>
```

Format-Specific Conventions

YAML Conventions

1. **Indentation**: Always use **2 spaces** per level—no tabs.
2. **Strings & Quoting**: URLs and simple keys rarely need quotes. If a string contains `:`, `#`, leading/trailing whitespace or special characters, wrap it in double quotes.
3. **Comments**: Use `#` for comments—YAML will ignore everything after it on the line.

JSON Conventions

1. **No Comments**: JSON doesn't support comments.
2. **Strict Syntax**: All strings must be quoted, no trailing commas allowed.
3. **Escaping**: Use `\"` for quotes within strings, `\\` for backslashes.

TOML Conventions

1. **Comments**: Use `#` for comments like YAML.
2. **Tables**: Use `[section]` for objects, `[[array]]` for arrays of objects.
3. **Strings**: Basic strings use double quotes, literal strings use single quotes.

XML Conventions

1. **Comments**: Use `<!-- comment -->` for comments.
2. **Attributes vs Elements**: Use `path` attribute for directory keys, elements for values.
3. **Escaping**: Use `&amp;` for `&`, `&lt;` for `<`, `&gt;` for `>`, `&quot;` for `"`.
4. **Boolean Values**: Use `true` / `false` as text content.
5. **Arrays**: Repeat element names for array items (e.g., multiple `<source>` elements).

---

1. `output_structure`: Pre-Create Directories

This block lets you declare **named output folders** you want created before any downloads start. It's purely "mkdir-style" bookkeeping.

YAML:

```yaml
output_structure:
  "#/images":
    description: "Artwork & screens"
    create: true
  "#/archives/raw":
    description: "Raw ZIPs for diagnostics"
    create: true
  "#/cache":
```

```
    description: "Caching internal files"
    create: false
```

JSON:

```
{
  "output_structure": {
    "#/images": {
      "description": "Artwork & screens",
      "create": true
    },
    "#/archives/raw": {
      "description": "Raw ZIPs for diagnostics",
      "create": true
    },
    "#/cache": {
      "description": "Caching internal files",
      "create": false
    }
  }
}
```

TOML:

```
[output_structure."#/images"]
description = "Artwork & screens"
create = true

[output_structure."#/archives/raw"]
description = "Raw ZIPs for diagnostics"
create = true

[output_structure."#/cache"]
description = "Caching internal files"
create = false
```

XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <output_structure>
    <dir path="#/images">
      <description>Artwork &amp;amp; screens</description>
      <create>true</create>
    </dir>
    <dir path="#/archives/raw">
      <description>Raw ZIPs for diagnostics</description>
      <create>true</create>
    </dir>
    <dir path="#/cache">
      <description>Caching internal files</description>
      <create>false</create>
    </dir>
```

```
  </output_structure>
</root>
```

- **Key syntax**:
  - `#` will be replaced by the CLI's `--output` directory (e.g. if you pass `--output=build`, `"#/images"` → `build/images`).
  - You can nest arbitrarily: `"#/foo/bar/baz"`.
- **Fields**:
  - `description` *(string)*: for your logs & team reference.
  - `create` *(boolean)*: if `true`, Qwikbrew runs `fs::create_dir_all()` here.
- **When to use**:
  - Guarantee folder existence for subsequent extraction.
  - Document your output layout for teammates.

---

2. `zip_sources`: Download & Extract ZIP Archives

Each entry in `zip_sources` follows this full schema:

YAML:

```yaml
zip_sources:
  - url: "https://example.com/archive.zip"        # (required) download link
    name: "Example Archive"                       # (required) identifier
    target_dir: "#/data"                         # where to extract; default "#"
    include_files: ["README.md", "LICENSE"]      # exact filenames to whitelist
    exclude_files: ["debug.log", "secret.txt"]   # exact filenames to blacklist
    include_patterns: ["*.so", "bin/*"]          # glob patterns to whitelist
    exclude_patterns: ["*_test.rs", "old_*"]     # glob patterns to blacklist
    preserve_structure: true                      # keep internal directory tree?
```

JSON:

```json
{
  "zip_sources": [
    {
      "url": "https://example.com/archive.zip",
      "name": "Example Archive",
      "target_dir": "#/data",
      "include_files": ["README.md", "LICENSE"],
      "exclude_files": ["debug.log", "secret.txt"],
      "include_patterns": ["*.so", "bin/*"],
      "exclude_patterns": ["*_test.rs", "old_*"],
      "preserve_structure": true
    }
  ]
}
```

TOML:

```toml
[[zip_sources]]
url = "https://example.com/archive.zip"
name = "Example Archive"
target_dir = "#/data"
include_files = ["README.md", "LICENSE"]
exclude_files = ["debug.log", "secret.txt"]
include_patterns = ["*.so", "bin/*"]
exclude_patterns = ["*_test.rs", "old_*"]
preserve_structure = true
```

XML:

```xml
<zip_sources>
  <source>
    <url>https://example.com/archive.zip</url>
    <name>Example Archive</name>
    <target_dir>#/data</target_dir>
    <include_files>
      <file>README.md</file>
      <file>LICENSE</file>
    </include_files>
    <exclude_files>
      <file>debug.log</file>
      <file>secret.txt</file>
    </exclude_files>
    <include_patterns>
      <pattern>*.so</pattern>
      <pattern>bin/*</pattern>
    </include_patterns>
    <exclude_patterns>
      <pattern>*_test.rs</pattern>
      <pattern>old_*</pattern>
    </exclude_patterns>
    <preserve_structure>true</preserve_structure>
  </source>
</zip_sources>
```

2.1 Extraction Flow & Precedence

1. **Download** via `reqwest::get()`.
2. **Iterate** over every entry in the ZIP.
3. **Skip** if `file.is_dir()`.
4. **Exclude checks** (highest priority):
    - If `exclude_files` contains the filename → drop.
    - If any `exclude_patterns` match the filename or path → drop.
5. **Include checks** (only if enabled):
    - If `include_files` is non-empty → file must appear there.
    - If `include_patterns` is non-empty → file must match at least one glob.
6. **Extract** to the computed path, creating parents as needed.

2.2 Field Reference

Field	Type	Default	Example & Notes
`url`	`string`	**required**	`https://…/project-v1.zip`
`name`	`string`	**required**	`"Project v1.0 Release"`
`target_dir`	`string`	`"#"`	`"#/libs/project"` → extracts into `output/libs/project`
`include_files`	`[string]`	`[]`	`["README.md","LICENSE"]`
`exclude_files`	`[string]`	`[]`	`["debug.log","secret.txt"]`
`include_patterns`	`[glob]`	`[]`	`["*.so","bin/*"]`
`exclude_patterns`	`[glob]`	`[]`	`["*_test.rs","old_*","docs/*.pdf"]`
`preserve_structure`	`boolean`	`false`	`true` → keeps subfolders (`src/foo.rs` stays in `src/`, not flattened`)

---

3. `file_sources`: One-Off File Downloads

Use `file_sources` to pull single files (configs, scripts, binaries):

YAML:

```yaml
file_sources:
  - url: "https://example.com/scripts/deploy.sh"
    target_path: "#/bin"
    rename_to: "qwik-deploy"
  - url: "https://example.com/configs/app.yaml"
    target_path: "#"
```

JSON:

```json
{
  "file_sources": [
    {
      "url": "https://example.com/scripts/deploy.sh",
      "target_path": "#/bin",
      "rename_to": "qwik-deploy"
    },
    {
      "url": "https://example.com/configs/app.yaml",
      "target_path": "#"
    }
  ]
}
```

TOML:

```toml
[[file_sources]]
url = "https://example.com/scripts/deploy.sh"
target_path = "#/bin"
rename_to = "qwik-deploy"
```

```toml
[[file_sources]]
url = "https://example.com/configs/app.yaml"
target_path = "#"
```

XML:

```xml
<file_sources>
  <source>
    <url>https://example.com/scripts/deploy.sh</url>
    <target_path>#/bin</target_path>
    <rename_to>qwik-deploy</rename_to>
  </source>
  <source>
    <url>https://example.com/configs/app.yaml</url>
    <target_path>#</target_path>
  </source>
</file_sources>
```

Field	Type	Default	Behavior
`url`	`string`	**required**	Full HTTP(S) URL to fetch.
`target_path`	`string`	`"#"`	Template for directory or full path; `#` means "root".
`rename_to`	`string?`	*none*	If provided, replaces the filename portion after path substitution.

---

4. Glob Patterns Deep Dive

Qwikbrew's simple glob rules operate on the **file name** (or full **relative path** when preserving structure):

Pattern	Matches
`*`	everything
`*.ext`	any name ending in `.ext`
`prefix*`	any name starting with `prefix`
`*suffix`	any name ending with `suffix`
`?`	exactly one arbitrary character
`docs/*`	any file directly under `docs/`
`docs/**/*.md` (not supported)	*recursive globs* are **not** yet supported—only single-segment `*`.

---

5. Complete Multi-Format Example

Here's a realistic configuration shown in all four formats:

YAML Version

```yaml
# Node.js deployment configuration
output_structure:
  "#/logs":
    description: "Runtime logs"
    create: true
  "#/temp":
    description: "Temp files"
    create: true

zip_sources:
  - url: "https://nodejs.org/dist/v18.14.0/node-v18.14.0-win-x64.zip"
    name: "NodeJS v18.14.0 Win64"
    target_dir: "#/bin"
    include_patterns:
      - "*.exe"
      - "*.dll"
    exclude_patterns:
      - "nodevars.bat"
    preserve_structure: false

file_sources:
  - url: "https://example.com/app/config.json"
    target_path: "#/conf"
    rename_to: "app-config.json"
```

JSON Version

```json
{
  "output_structure": {
    "#/logs": {
      "description": "Runtime logs",
      "create": true
    },
    "#/temp": {
      "description": "Temp files",
      "create": true
    }
  },
  "zip_sources": [
    {
      "url": "https://nodejs.org/dist/v18.14.0/node-v18.14.0-win-x64.zip",
      "name": "NodeJS v18.14.0 Win64",
      "target_dir": "#/bin",
      "include_patterns": ["*.exe", "*.dll"],
      "exclude_patterns": ["nodevars.bat"],
      "preserve_structure": false
    }
  ],
  "file_sources": [
```

```
    {
      "url": "https://example.com/app/config.json",
      "target_path": "#/conf",
      "rename_to": "app-config.json"
    }
  ]
}
```

TOML Version

```toml
# Node.js deployment configuration

[output_structure."#/logs"]
description = "Runtime logs"
create = true

[output_structure."#/temp"]
description = "Temp files"
create = true

[[zip_sources]]
url = "https://nodejs.org/dist/v18.14.0/node-v18.14.0-win-x64.zip"
name = "NodeJS v18.14.0 Win64"
target_dir = "#/bin"
include_patterns = ["*.exe", "*.dll"]
exclude_patterns = ["nodevars.bat"]
preserve_structure = false

[[file_sources]]
url = "https://example.com/app/config.json"
target_path = "#/conf"
rename_to = "app-config.json"
```

XML Version

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Node.js deployment configuration -->
<root>
  <output_structure>
    <dir path="#/logs">
      <description>Runtime logs</description>
      <create>true</create>
    </dir>
    <dir path="#/temp">
      <description>Temp files</description>
      <create>true</create>
    </dir>
  </output_structure>

  <zip_sources>
    <source>
```

```
      <url>https://nodejs.org/dist/v18.14.0/node-v18.14.0-win-x64.zip</url>
      <name>NodeJS v18.14.0 Win64</name>
      <target_dir>#/bin</target_dir>
      <include_patterns>
        <pattern>*.exe</pattern>
        <pattern>*.dll</pattern>
      </include_patterns>
      <exclude_patterns>
        <pattern>nodevars.bat</pattern>
      </exclude_patterns>
      <preserve_structure>false</preserve_structure>
    </source>
  </zip_sources>

  <file_sources>
    <source>
      <url>https://example.com/app/config.json</url>
      <target_path>#/conf</target_path>
      <rename_to>app-config.json</rename_to>
    </source>
  </file_sources>
</root>
```

---

6. Validation & Troubleshooting

Format-Specific Validation

- **YAML**: Use `yamllint config.yaml` to catch syntax issues.
- **JSON**: Use `jsonlint config.json` or any JSON validator.
- **TOML**: Use `toml-validator config.toml` or similar tools.
- **XML**: Use `xmllint config.xml` or any XML validator to check well-formedness.

Common Issues

- **Serde errors**: Qwikbrew will print which field/line was invalid.
- **Empty extractions**: Check your `include_*` vs. `exclude_*` overlap.
- **Format detection**: Use `--override-type-detection` if auto-detection fails.
- **XML escaping**: Remember to escape `&`, `<`, `>` in XML content.
- **Verbose logging**: Qwikbrew prints each "Created directory" and "Extracted" line.

---

7. Tips & Best Practices

Format Selection

- **YAML**: Best for human readability, comments, and complex configurations.
- **JSON**: Best for programmatic generation and web API integration.
- **TOML**: Best for simple, clear configurations with good readability.
- **XML**: Best for structured data exchange, validation with schemas, and integration with XML-based systems.

General Tips

- **Comments**: Use them liberally in YAML, TOML, and XML (JSON doesn't support comments).

- **Validation**: Always validate your config files before deployment.
- **Version pinning**: Include exact URLs with version numbers for reproducible builds.
- **Testing**: Use `--override-type-detection` to test the same config in different formats.
- **Migration**: You can convert between formats using online converters or tools.
- **XML namespaces**: Not currently supported—keep XML simple and flat.

---

8. CLI Reference

Basic Usage

```
# Auto-detect format
qwikbrew extract -c config.yaml

# Specify output directory
qwikbrew extract -c config.json -o build

# Override format detection
qwikbrew extract -c myconfig.txt --override-type-detection=toml

# Use XML format
qwikbrew extract -c config.xml
```

Format Override Options

- `yaml` or `yml` - Parse as YAML
- `json` - Parse as JSON
- `toml` - Parse as TOML
- `xml` - Parse as XML
- Case insensitive: `YAML`, `Json`, `Toml`, `XML` all work

With this comprehensive guide, you can now craft **robust**, **readable**, and **precise** configuration files for Qwikbrew in any supported format—whether you're orchestrating a handful of file grabs or automating a multi-step, multi-ZIP build pipeline. The addition of XML support in Q3.1.1-D1 provides even more flexibility for integration with existing XML-based workflows and systems. Happy brewing!