Stride Configuration Spec

Version: S4.0.0-D1

An **in-depth reference** for authoring configuration files for **Stride**, 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

Stride now supports multiple configuration formats:

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

Format Detection

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

```
# Auto-detected formats
stride extract -c config.yaml    # YAML
stride extract -c config.json    # JSON
stride extract -c config.toml    # TOML
stride 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
stride extract -c myconfig.txt --override-type-detection=yaml

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

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

# Force XML parsing
stride 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

archive_sources:
  - url: "https://example.com/archive.zip"
    name: "Example Archive"
    type: "zip"
    target_dir: "#/data"
    include_directories: ["src", "docs"]
```

JSON Example

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

TOML Example

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

[[archive_sources]]
url = "https://example.com/archive.zip"
name = "Example Archive"
type = "zip"
target_dir = "#/data"
include_directories = ["src", "docs"]
```

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>
  <archive_sources>
    <source>
      <url>https://example.com/archive.zip</url>
      <name>Example Archive</name>
      <type>zip</type>
      <target_dir>#/data</target_dir>
      <include_directories>
        <directory>src</directory>
        <directory>docs</directory>
      </include_directories>
    </source>
  </archive_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 archives for diagnostics"
    create: true
  "#/cache":
    description: "Caching internal files"
    create: false
```

JSON:

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

TOML:

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

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

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

XML:

```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 archives 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`, Stride runs `fs::create_dir_all()` here.
- **When to use**:
  - Guarantee folder existence for subsequent extraction.
  - Document your output layout for teammates.

---

2. `archive_sources`: Download & Extract Archives

With Stride 4.0.0, `zip_sources` is now `archive_sources`. An archive source now requires a field called `type` which supports `zip`, `tar`, `gz` and can be stacked (e.g., `"zip"` or `"tar.gz"`). Each entry follows this schema:

YAML:

```yaml
archive_sources:
  - url: "https://example.com/archive.zip" # (required)
    name: "Example Archive"                # (required)
    type: "zip"                            # supports zip, tar, gz, or combinations
like "tar.gz"
    target_dir: "#/data"                   # default "#"
    include_files: ["README.md"]           # optional
    exclude_files: []                      # optional
    include_patterns: []                    # optional
    exclude_patterns: []                    # optional
    include_directories: []                 # optional
    exclude_directories: []                 # optional
    preserve_structure: true                # default: false
    strip_root_folder: "archive-main"      # optional; remove this top-level folder
when extracting
```

JSON:

```
{
  "archive_sources": [
    {
      "url": "https://example.com/archive.zip",
      "name": "Example Archive",
      "type": "zip",
      "target_dir": "#/data",
      "preserve_structure": true,
      "strip_root_folder": "archive-main"
    }
  ]
}
```

TOML:

```
[[archive_sources]]
url = "https://example.com/archive.zip"
name = "Example Archive"
type = "zip"
target_dir = "#/data"
preserve_structure = true
strip_root_folder = "archive-main"
```

XML:

```
<archive_sources>
  <source>
    <url>https://example.com/archive.zip</url>
    <name>Example Archive</name>
    <type>zip</type>
    <target_dir>#/data</target_dir>
    <preserve_structure>true</preserve_structure>
    <strip_root_folder>archive-main</strip_root_folder>
  </source>
</archive_sources>
```

Field	Type	Default	Notes
`url`	`string`	**required**	Download link.
`name`	`string`	**required**	Identifier for logs (e.g. "Example Archive").
`type`	`string`	**required**	Archive type: `zip`, `tar`, `gz`, or stacked (e.g., `"tar.gz"`).
`target_dir`	`string`	`"#"`	Destination directory template.
`include_files`	`[string]`	`[]`	Exact filenames to whitelist.
`exclude_files`	`[string]`	`[]`	Exact filenames to blacklist.
`include_patterns`	`[glob]`	`[]`	Glob patterns to whitelist.
`exclude_patterns`	`[glob]`	`[]`	Glob patterns to blacklist.

`include_directories`	`[string]`	`[]`	Exact directory paths to whitelist.
`exclude_directories`	`[string]`	`[]`	Exact directory paths to blacklist.
`preserve_structure`	`boolean`	`false`	`true` preserves internal folder hierarchy, `false` flattens all files.
`strip_root_folder`	`string?`	*none*	Name of top-level folder to strip (e.g. GitHub `repo-main`). Optional.