Formats & Projects

Session 4

Mickaël CANOUIL, Ph.D.[1*]

Saturday, the 7th of February, 2026

Affiliations

[1] Independent Contractor, Lille, France

ORCID

Mickaël CANOUIL, Ph.D.: 0000-0002-3396-4549

[*]Corresponding author: pro@mickael.canouil.dev

Table of Contents

Table of Contents

Table of Contents

1. Session Overview

1.1. Session 4: Formats & Projects

- Quarto Projects
  - Project types and architecture: default, website, blog, manuscript, book.
  - Project type decision framework based on publishing goals.
  - Understanding project structure and configuration files.
- Multiple Formats
  - Multi-format output optimisation with format-specific options.
  - Conditional content by format using content-visible/hidden classes.
  - Cross-format resource management and figure optimisation.
- Advanced Project Features
  - Shared metadata and configuration with _metadata.yml.
  - Pre and post-render scripts for workflow automation.
  - Project profiles for different deployment scenarios.
  - Environment variables and resource management.

1.2. Session Objectives

This session explores Quarto's project architecture and format ecosystem, covering project types, multi-format optimisation, and template systems for scalable workflows.

1.3. Learning Objectives

By the end of this session, participants will be able to:

- Select appropriate project types for different publishing scenarios.
- Configure multi-format outputs with format-specific optimisations.
- Use template systems for rapid project initiation.
- Implement collaborative workflows with proper project structure.
- Understand format capabilities and limitations.

2. Quarto Projects

2.1. Project Types and Architecture

Recall the project types introduced in Session 1. Here we examine their configuration and architecture in more detail.

```bash
quarto create project
```

```
? Type
❯ default
  website
  blog
  manuscript
  book
  confluence
```

2.2. Project Type Decision Framework

Choose your project type based on your publishing goals:

Scenario	Project Type	Key Features
Multiple unrelated documents	`default`	Flexible structure, shared metadata
Single research paper	`manuscript`	Academic formatting, citations, submission-ready
Public documentation/blog	`website` / `blog`	Navigation, search, responsive design
Long-form content	`book`	Cross-chapter references, multiple outputs

2.3. Default Project Structure

The most flexible option for diverse content:

```yaml
project:
  title: "Default"
```

2.4. Website Projects

Optimised for web publishing:

```yaml
project:
  type: website

website:
  title: "Website"
  navbar:
    left:
      - href: index.qmd
        text: Home
      - about.qmd

format:
  html:
    theme:
      - cosmo
      - brand
    css: styles.css
    toc: true
```

2.5. Blog Projects (website + listing)

Designed for regular posts and updates:

```yaml
project:
  type: website

website:
  description: "A blog built with Quarto"
  site-url: https://your-website-url.example.com # you must change this
appropriately for RSS feeds to work properly
  title: "Blog"
  navbar:
    right:
      - about.qmd
      - icon: github
        href: https://github.com/
      - icon: bluesky
        href: https://bsky.app/
format:
```

```yaml
  html:
    theme:
      - cosmo
      - brand
    css: styles.css
```

2.6. Manuscript Projects

Optimised for academic publishing:

```yaml
project:
  type: manuscript

manuscript:
  article: index.qmd

format:
  html:
    comments:
      hypothesis: true
  docx: default
  jats: default

  # (other formats)
  # pdf: default

execute:
  freeze: true
```

2.7. Book Projects

For comprehensive, structured content:

```yaml
project:
  type: book

book:
  title: "Book"
  author: "Norah Jones"
```

```yaml
  date: "2/7/2026"
  chapters:
    - index.qmd
    - intro.qmd
    - summary.qmd
    - references.qmd

bibliography: references.bib

format:
  html:
    theme:
      - cosmo
      - brand
  pdf:
    documentclass: scrreprt
```


3. Multiple Formats

3.1. Multi-Format Output Optimisation

Balance capability with compatibility by configuring format-specific options that leverage each output type's strengths.

```yaml
title: "Multi-Format Document"
fig-width: 6
fig-height: 4
format:
  html:
    theme: cosmo
    code-fold: true
    code-tools: true                    (1)
    toc: true
    fig-width: 8
    fig-height: 6
  pdf:
    documentclass: article
    geometry: margin=1in
    fontsize: 11pt
    colorlinks: true
  typst:
    brand-mode: dark
  revealjs:
    brand-mode: dark
    fig-width: 10
    fig-height: 6                       (2)
```

(1) Setting format-specific options.

(2) Setting shared and format-specific options.

3.2. Conditional Content by Format

Control what content appears in different output formats.

3.2.1. Key Classes

- `.content-visible` - Show content only for specific formats.
- `.content-hidden` - Hide content from specific formats.

3.2.2. Conditions

- `when-format` / `unless-format` - Control visibility based on output format.
- `when-meta` / `unless-meta` - Control visibility based on metadata.
- `when-profile` / `unless-profile` - Control visibility based on profile.

3.2.3. Syntax

- ```MARKDOWN
  ::: {.content-visible when-format="html"}
  HTML-only content
  :::
  ```

- ```MARKDOWN
  [Typst/PDF only text]{.content-visible when-format="typst"}
  ```

- ```MARKDOWN
  ```{.python .content-visible when-format="html"}
  print("Only shows in HTML")
  ```
  ```

- ```MARKDOWN
  ::: {.content-visible when-format="html"}
  ```{{python}}
  print("Only shows in HTML but is evaluated in all formats")
  ```
  :::
  ```

3.3. Cross-Format Resource Management

Optimise images and assets for different outputs:

```YAML
format:
  html:
    fig-format: svg      # Vector graphics for web
    fig-dpi: 150
  pdf:
    fig-format: pdf      # Vector graphics for print
    fig-dpi: 300
  typst:
    fig-format: png      # Raster for compatibility
    fig-dpi: 150
```


4. Advanced Project Features

4.1. Shared Metadata and Configuration

Use `_metadata.yml` for common settings to be shared by documents in a directory:

subdirectory/_metadata.yml

```yaml
format:
  html:
    theme: custom
execute:
  echo: false
  warning: false
```

4.2. Resource Management

Organise assets that needs to be copied to the output directory:

_quarto.yml

```yaml
project:
  resources:
    - "data/*.csv"
    - "images/"
    - "!temp/"      # Exclude temporary files
```

4.3. Pre and Post Render Scripts

Automate workflows before and after rendering.

4.3.1. Configuration

_quarto.yml

```yaml
project:
  pre-render:
    - data-prep.py
    - generate-content.R
  post-render:
    - cleanup.sh
    - deploy.ts
```

4.3.2. Key Features

- Scripts run from project root directory.
- Re-computes metadata after pre-render (can generate new files).
- Supports shell commands, Python, R, TypeScript, Lua.

4.3.3. Environment Variables Available

- `QUARTO_PROJECT_RENDER_ALL` - "1" if rendering entire project.
- `QUARTO_PROJECT_OUTPUT_DIR` - Output directory path.
- `QUARTO_PROJECT_INPUT_FILES` - List of input files (pre-render).
- `QUARTO_PROJECT_OUTPUT_FILES` - List of output files (post-render).

4.3.4. Conditional Execution Example

PYTHON

```python
# Only run expensive operations on full renders
import os
if os.environ.get("QUARTO_PROJECT_RENDER_ALL") == "1":
    # Run expensive data processing
    process_large_dataset()
```

4.4. Environment Variables

Manage project environment variables with files and profiles:

- `_environment` - Default environment variables for all renders.
- `_environment.local` - Local overrides (auto-ignored by Git).
- `_environment-{profile}` - Profile-specific variables.
- `_environment.required` - Documentation of required variables.

4.5. Variables

Insert dynamic content with shortcodes:

- `{{< var variable_name >}}` - From `_variables.yml` file.
- `{{< meta field_name >}}` - From document YAML metadata.
- `{{< env ENV_VARIABLE >}}` - From environment variables.

4.5.1. Markdown

MARKDOWN

```markdown
- {{< meta title >}}
- {{< meta subtitle >}}
```

4.5.2. Output

- Formats & Projects
- Session 4

4.6. Project Profiles

Adapt projects for different scenarios with configuration variants.

4.6.1. What Are Profiles?

- Different configurations for the same project (production, development, basic/advanced versions).

4.6.2. Configuration Files

- `_quarto.yml` - Base configuration.
- `_quarto-{profile}.yml` - Profile-specific config that merges with base.
- `_environment-{profile}` - Profile-specific environment variables.

4.6.3. Activation

BASH

```bash
# Environment variable
export QUARTO_PROFILE=production
quarto render

# Command line
quarto render --profile production

# Multiple profiles
quarto render --profile production,advanced
```

4.6.4. Content Targeting

MARKDOWN

```markdown
::: {.content-visible when-profile="advanced"}
Advanced content only
:::

::: {.content-hidden unless-profile="basic"}
Hidden for basic profile
:::
```

4.6.5. Profile Groups & Defaults

YAML

```yaml
profile:
  group: [basic, advanced]
  default: development
```

5. Hands-On Exercise: Building Your First Project

5.1. Exercise Overview

Objective: Transform your computational portfolio from Session 3 into a structured project and explore multiple output formats.

Example Code: Exercises

```bash
tar -xzf "04-exercises.tar.gz" -C "04-formats-projects"
```

5.2. Part 1: Create a Project Structure

Convert your portfolio into a project:

5.2.1. Website Project

Create a `_quarto.yml` file:

```yaml
project:
 type: website

website:
 title: "My Learning Portfolio"
 navbar:
   left:
     - href: index.qmd
       text: Home
     - about.qmd

format:
 html:
   theme: cosmo
   toc: true
```

5.2.2. Default Project

Create a `_quarto.yml` file:

```yaml
project:
 title: "My Portfolio Collection"

format:
 html:
   theme: cosmo
```



```
    toc: true
  typst: default
```

5.2.3. Book Project

Create a `_quarto.yml` file:

```yaml
project:
  type: book

book:
  title: "My Quarto Journey"
  chapters:
    - index.qmd
    - portfolio.qmd
    - reflection.qmd

format:
  html:
    theme: cosmo
```

5.3. Part 2: Multi-Format Configuration

Add multiple output formats to your project:

5.3.1. Basic Multi-Format

Add to your `_quarto.yml`:

```yaml
format:
  html:
    theme: cosmo
    code-fold: true
  typst:
    margin:
      top: 2cm
      bottom: 2cm
  docx: default
```

5.3.2. Advanced Configuration

Add format-specific options:

```yaml
format:
 html:
   theme: cosmo
   code-fold: true
   fig-width: 8
   fig-height: 6
 typst:
   fig-width: 6
   fig-height: 4
 pdf:
   geometry:
     - margin=1in
   fontsize: 11pt
```

5.3.3. Presentation Format

Add slides to your outputs:

```yaml
format:
 html: default
 revealjs:
   theme: dark
   fig-width: 10
   fig-height: 6
```

5.4. Part 3: Conditional Content

Add format-specific content to your documents:

5.4.1. Format-Specific Sections

Add to your content:

```markdown
::: {.content-visible when-format="html"}
This interactive content only appears in HTML output.
:::

::: {.content-visible when-format="typst"}
This formatted text appears only in Typst/PDF output.
:::

::: {.content-hidden when-format="revealjs"}
```



```
This detailed content is hidden in presentations.
:::
```

5.4.2. Code Visibility by Format

Control code display:

MARKDOWN

````
```{{python .content-visible when-format="html"}}
# Interactive code for web
import plotly.express as px
# Create interactive plot
```

```{{python .content-visible unless-format="html"}}
# Static plot for print formats
import matplotlib.pyplot as plt
# Create static plot
```
````

5.4.3. Format-Specific Messages

Add dynamic text:

MARKDOWN

```
[[View the interactive version online](link-to-html)]{.content-visible when-format="typst"}

[*This is the web version with interactive features*]{.content-visible when-format="html"}
```

5.5. Part 4: Test Multiple Formats

Render your project in different formats:

5.5.1. Single Format

Try rendering to specific formats:

BASH

```
quarto render --to html
quarto render --to typst
quarto render --to docx
```


5.5.2. All Formats

Render all configured formats:

```bash
quarto render
```

5.5.3. Preview Mode

Use preview for development:

```bash
quarto preview
```

5.6. Part 5: Add Project Features

Enhance your project with additional features:

5.6.1. Shared Metadata

Create `_metadata.yml`:

```yaml
author: "Your Name"
date: today
execute:
  echo: true
  warning: false
```

5.6.2. Resource Management

Add to `_quarto.yml`:

```yaml
project:
  resources:
    - "data/"
    - "images/"
    - "*.csv"
```

5.6.3. Environment Variables

Create `_environment` file:


```txt
QUARTO_PYTHON=/path/to/python
API_KEY=your-key-here
```

5.7. Success Criteria

☑ You've successfully completed the exercise if you can:

- Create a properly structured project with `_quarto.yml`.
- Configure at least two different output formats.
- Add format-specific content that shows/hides appropriately.
- Successfully render your project to multiple formats.
- Test that conditional content works correctly.