Code Window

Quarto Extension

Mickaël CANOUIL, *Ph.D.*

Table of contents

Explicit Filename

Code blocks with a `filename` attribute display a window header with the filename. The decoration style depends on the `style` option (default: `"macos"`).

fibonacci.py

```python
def fibonacci(n: int) -> int:
    """Calculate the nth Fibonacci number."""
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)
```

analysis.R

```r
# Load data and create summary
data <- read.csv("data.csv")
summary(data)
```

Auto-Generated Filename

With `auto-filename: true` (the default), code blocks without explicit filenames automatically show the language name in small-caps styling.

PYTHON

```python
def greet(name: str) -> str:
    """Return a greeting message."""
    return f"Hello, {name}!"
```

R

```r
# Create sample data
data <- data.frame(
  x = 1:10,
  y = rnorm(10)
)
summary(data)
```

BASH

```bash
#!/bin/bash
echo "Hello, World!"
```

Plain Code Block

Code blocks without a language are not affected by the extension.

```
This is a plain code block without any language specified.
No window decoration is applied here.
```

Disabled Auto-Filename

Per Block

Set `code-window-no-auto-filename="true"` on a single block to suppress the auto-generated filename without changing the global setting.

```
print("No auto-filename on this block")
```

Globally

To disable auto-generated filenames for the entire document, set `auto-filename: false` in the extension configuration. Only code blocks with an explicit `filename` attribute will display window decorations.

YAML

```yaml
extensions:
  code-window:
    auto-filename: false
```

Highlighted Lines

When a code block uses Quarto's `code-line-numbers` attribute, the line specification is shown as a chip next to the filename.

loader.py 1,4-5

```
import pandas as pd

def load(path):
    df = pd.read_csv(path)
    return df.dropna()
```

Use `code-window-lines` to override the displayed text without affecting Quarto's highlighting:

custom.py 2-3

```
def greet(name):
    message = f"Hello, {name}!"
    print(message)
```

Disable the chip globally with `lines-label: false` in the extension config.

Collapsible Code Windows (HTML)

Set `code-window-collapse` to wrap a code block in a `<details>` element. The title bar becomes the `<summary>`. This feature is HTML-only; Typst and PDF output is unaffected.

Closed by Default

hidden.py

```
print("Click the title bar to expand.")
print("Useful for long boilerplate.")
```

Open by Default

visible.py

```
print("Visible by default, can be collapsed.")
```

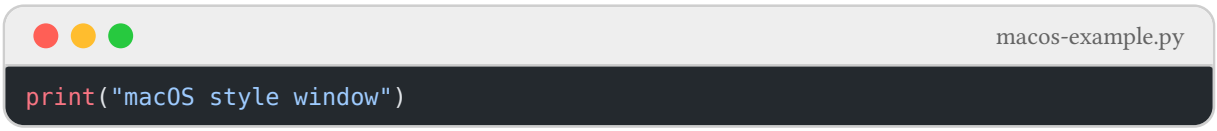
Apply collapsing to every code window by setting `collapse: closed` (or `open`) in the extension config.

Window Styles

Three decoration styles are available via the `style` option. The global style can be set in the document configuration. Individual blocks can override the style with the `code-window-style` attribute.

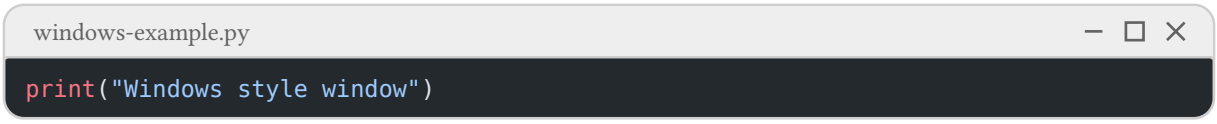
macOS Style (default)

Traffic light buttons on the left, filename on the right.



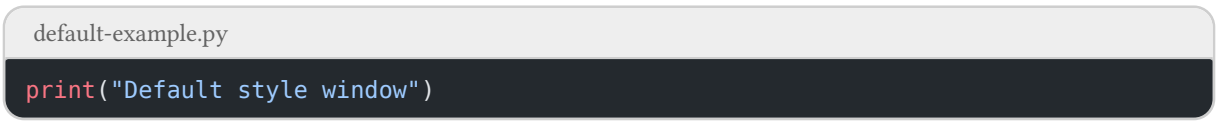
Windows Style

Minimise, maximise, and close buttons on the right, filename on the left.



Default Style

Plain filename on the left, no window decorations.



Code Annotations

> **Note**
>
> Typst code-annotations support and `filename` attribute handling are temporary hot-fixes. They will be removed once Quarto natively supports these features (see quarto-dev/quarto-cli#14170). The extension will then focus on **auto-filename** and **code-window-style** features.

Code annotations work standalone and together with code-window styling.

Annotations with Explicit Filename

annotated.py

```python
import pandas as pd                    ①

df = pd.read_csv("data.csv")           ②
summary = df.describe()                ③
```

① Import the pandas library.
② Load data from a CSV file.
③ Generate summary statistics.

Annotations with Auto-Filename

PYTHON

```python
def greet(name: str) -> str:           ①
    return f"Hello, {name}!"           ②

result = greet("World")                ③
```

① Define a function with type hints.
② Use an f-string for interpolation.
③ Call the function and store the result.

Annotations Spanning Multiple Lines

A single annotation number can appear on several consecutive lines. Only the first occurrence receives a back-label to avoid duplicates.

pipeline.py

```python
def process(data):                     ①
    cleaned = clean(data)              ①
    validated = validate(cleaned)      ①
    result = transform(validated)      ②
    return result                      ③
```

① Multi-step input preparation (cleaning and validation).
② Apply the main transformation.
③ Return the final result.

Annotations without Window Chrome

Set `code-window-enabled="false"` on a block to disable window chrome while keeping annotations.

```r
library(ggplot2)                                    # (1)
ggplot(mtcars) +                                    # (2)
  aes(x = mpg, y = hp) +                            # (3)
  geom_point()                                      # (4)
```

(1) Load the ggplot2 package.
(2) Initialise a plot with the mtcars dataset.
(3) Map aesthetics.
(4) Add a point geometry layer.

Configuration

Set the global style in the document front matter:

YAML

```yaml
extensions:
  code-window:
    enabled: true
    auto-filename: true
    style: "macos"
    wrapper: "code-window"
    hotfix:
      code-annotations: true
      skylighting: true
      typst-title: true
```

Each hotfix accepts either a boolean or a map with `enabled` and `quarto-version` keys for per-hotfix version thresholds:

YAML

```yaml
extensions:
  code-window:
    hotfix:
      code-annotations: true
      skylighting:
        enabled: false
      typst-title:
        quarto-version: "1.10.0"
```

Override per block with the `code-window-style` attribute:

MARKDOWN

````
```{.python filename="example.py" code-window-style="windows"}
print("Windows style for this block only")
```
````