Sphinx Touchbook Author Guide

Release 0.2.3-alpha

Dave Parillo

Jul 13, 2026

CONTENTS

CHAPTER

ONE

INTRODUCTION

`sphinx-touchbook` is a Sphinx extension project for authors who want interactive textbook pages without giving up the strengths of ordinary Sphinx documents. The goal is to let authors write semantic directives in reStructuredText, build static HTML for the web, and still produce useful non-interactive output for text and PDF-oriented builders.

The project starts from a simple premise: interactive textbook content should be compiled, not hand-written as fragile HTML fragments. Authors describe educational intent. The Sphinx extension turns that intent into semantic docutils nodes. Python generators render those nodes for each builder. In HTML, JavaScript components add interactive behavior to the generated custom elements.

1.1 Goals

The project is designed around a few practical goals:

- Keep author syntax concise and teachable.
- Preserve Sphinx features such as nested markup, code highlighting, cross references, and multiple builders.
- Generate accessible HTML that works before JavaScript runs.
- Make every component testable without requiring a complete textbook project.
- Keep interactive behavior reusable outside Sphinx when possible.
- Treat PDF, text, and no-JS HTML output as first-class fallbacks, not afterthoughts.

1.2 Architecture

Interactive textbook components use three layers:

1. Python Sphinx extension layer Roles, directives, domains, transforms, and docutils nodes parse author content and store semantic data. This layer does not implement browser behavior.
2. Python generator layer Builder-specific renderers turn semantic nodes into HTML, text, LaTeX-oriented output, or other builder output. In HTML, each directive emits exactly one custom element.
3. JavaScript component layer Custom elements implement browser behavior by hydrating the rendered HTML, its attributes, and its fallback content.

Some components may use optional stateless services for work that cannot reasonably happen in the browser, such as compiling code or evaluating symbolic math. Those services are not part of the required static publishing path.

1.3 Design philosophy

Each directive should describe an educational concept rather than a JavaScript library call. A data-structure animation directive should describe the operation sequence. A future renderer might use SVG, Canvas, JSAV, D3, or another library. The textbook source should not need to change simply because a rendering library changes.

The same rule applies to executable examples. The directive should describe editable, runnable source code. Whether the code runs through a local WebAssembly compiler, a hosted execution service, or a future backend is an implementation detail.

Custom elements are the HTML contract. A directive such as `tb-reveal` maps to one `<tb-reveal>` element. Controls, fallback content, and configuration belong inside that element. This keeps generated HTML understandable, testable, and reusable.

1.4 Progressive enhancement

Every component should start as useful static content. JavaScript may improve the experience, but it should not be the only way to reach essential content.

For example, `tb-reveal` emits a native `details` and `summary` fallback inside the custom element. Without JavaScript, readers can still open the disclosure. With JavaScript, the fallback is replaced with inline or modal behavior.

This approach also supports non-HTML builders. A reveal block can become labeled static content in text output and a readable block in PDF-oriented output. The reader loses the interaction, but not the information.

1.5 Testing approach

The test strategy follows the same layer boundaries:

- Sphinx directive tests check parsing, options, diagnostics, and semantic node fields.
- Python generator tests check custom element output, fallback content, assets, and non-HTML builder behavior.
- JavaScript component tests check browser behavior, accessibility state, keyboard behavior, and service failure handling.
- Documentation builds act as end-to-end tests for author-facing examples.

Routine tests should use small temporary Sphinx projects. They should not require a real textbook repository.

1.6 Author guide structure

Each component page documents the author syntax, options, accessibility behavior, fallback behavior, and examples. The examples are real Sphinx source that builds as part of this guide, so the guide is both a manual for authors and a regression test for the project.

CHAPTER

TWO

GENERAL SYNTAX

All directives start with `..`, then a single space, followed by the name of the directive, then `::`.

Many directives also accept options. Options are indented below the directive line and begin with `::`

Source

```
.. tb-reveal::
   :showlabel: Show answer
   :hidelabel: Hide answer

   The answer is 42.
```

Rendered

Show answer

The answer is 42.

Container directives can contain other directives:

Source

```
.. tb-group::

   .. tb-tab:: First tab

      Content for the first tab.

   .. tb-tab:: Useful forms

      Here is a useful summation, along with the closed-form solution:

      .. math::

         \sum_{k = 1}^{n} k = \frac{n (n+1)}{2}.
```

Rendered

First tab

Content for the first tab.

Useful forms

Here is a useful summation, along with the closed-form solution:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}.$$

2.1 Common options

Every Touchbook directive accepts the optional `name` and `class` parameters as described in Docutils common option. If `name` is omitted, an ID is automatically generated based on the current document and node position. This ID is deterministic, but not stable: if your document changes the generated name may change. Use an explicit `name` when you want a stable human-readable target for references, tests, or custom integration.

```
.. tb-reveal::
   :name: optional-name

   Content that can be revealed.
```

The `class` parameter takes a space-separated list of class names. HTML output attaches those classes to the directive's root `tb-*` element.

```
.. tb-reveal::
   :class: review-note compact

   Content that can be styled with project CSS.
```

```
.. tb-code:: python
   :class: highlight-override

   print("Hello")
```

CHAPTER

THREE

TOUCHBOOK DIRECTIVES DOCUMENTATION

Every Touchbook directive has a single purpose. Each is detailed below, including:

- What each directive allows you to create
- The syntax for using each directive and parameter
- Examples, or links to examples, of how instructors can use these directives in interactive textbook work
- Available additional developer documentation or notes

Custom Touchbook directives exist in the following general categories. Implemented directives link to detailed pages. Planned directives are listed as plain text until they exist.

Categories	Directives
Working with Code	• *tb-code* • *tb-file*
Containers	• *tb-group* • *tb-reveal* • *tb-video*
Assessments	• *tb-blank* • *tb-choice* • *tb-click* • *tb-formula* • *tb-match* • *tb-micro-parsons* • *tb-order* • *tb-parsons*

3.1 tb-blank

The `tb-blank` directive creates a fill-in-the-blank question. Authors place one or more `{{blank}}` markers in normal Sphinx content. Nested `tb-answer` blocks define accepted answers, targeted hints for common wrong answers, and fallback incorrect feedback.

3.1.1 Synopsis

The general format of the `tb-blank` directive is:

```
.. tb-blank::
   :optional parameter: value

   + --- Prompt area ---
   |
   | question text and optional Sphinx content with {{blank}} markers
   |
   + --- Answer area ---
   |
   | .. tb-answer:: optional-blank-id
   |    :match: accepted answer
   |    :feedback: correct feedback
   |    :hint: known wrong answer; targeted feedback
   |    :incorrect: fallback incorrect feedback
   |
   + -----------------
```

3.1.2 Options

case-sensitive
: `Flag`. Optional. Require exact case when comparing submitted answers. By default, matching is case-insensitive.

class
: `String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

name
: `String`. Optional. Sphinx reference name for this fill-in-the-blank question. See *Common options* for details.

preserve-whitespace
: `Flag`. Optional. Preserve leading and trailing whitespace when comparing submitted answers. By default, submitted answers and expected answers are trimmed.

3.1.3 Answer Options

Nested `tb-answer` blocks support these options:

feedback
: `String`. Optional. Feedback shown when a submitted answer matches one of the accepted answers.

hint
: `String`. Optional. Repeatable. Known wrong answer and targeted feedback separated by the first semicolon. For example, `:hint:  7; Check the assignment in the else block.`.

incorrect
: `String`. Optional. Feedback shown when the submitted answer does not match an accepted answer or a known wrong answer.

match
: `String`. Required. Repeatable. Accepted answer. Matching is case-insensitive and trims leading and trailing whitespace unless directive options override those defaults.

regex
: `Flag`. Optional. Treat `match` values as regular expressions.

3.1.4 Accessibility behavior

HTML uses native text inputs and a native button. Result text uses a status region so assistive technology can announce feedback after checking.

3.1.5 Fallback behavior

HTML without JavaScript renders the prompt and input fields. Text and PDF-oriented builders render each blank as an underline. Answers and feedback are not rendered in fallback output.

3.1.6 Examples

Example 1: Single blank

Source

```
.. tb-blank::

   The capital of France is {{blank}}.

   .. tb-answer::
      :match: Paris
      :feedback: Correct.
      :incorrect: Try again.
```

Rendered

Fill in the blank

The capital of France is _____.

Example 2: Code reasoning with targeted hints

Source

```
.. tb-blank::

   .. code-block:: c

      int main(void) {
        int a = 7;
        int b = 4;

        if (a<=b) {
          a = 99;
        } else {
          int t = a;
          a = b;
          b = t;
        }
        return a;
      }

   What value is returned from main? {{blank}}
```

(continues on next page)

(continued from previous page)

```
   .. tb-answer::
      :match: 4
      :match: 4.0
      :feedback: Correct.
      :hint: 7; No, because the variable a is modified in the else block.
      :hint: 99; No. Since a is greater than b, line 6 is never executed.
      :incorrect: Sorry, no. What is happening in the else block?
```

Rendered

Fill in the blank

```
int main(void) {
  int a = 7;
  int b = 4;

  if (a<=b) {
    a = 99;
  } else {
    int t = a;
    a = b;
    b = t;
  }
  return a;
}
```

What value is returned from main? _____

3.2 tb-choice

The `tb-choice` directive creates a multiple choice or multiple answer question. Use one top-level bullet list for answer choices. For compact questions, mark each answer with `[x]` for correct or `[ ]` for incorrect. Any content after the first answer paragraph becomes feedback.

For complex multi-block answer content, use a nested feedback bullet. Mark correct feedback with `+` and incorrect feedback with `-`.

If exactly one answer is marked correct, HTML uses radio buttons. If more than one answer is marked correct, HTML uses checkboxes.

3.2.1 Synopsis

The `tb-choice` directive allows questions and feedback defined in one of two formats: 'compact' and 'nested list'. The compact format looks like this:

```
.. tb-choice::
   :optional parameter: value

   + --- Prompt area ---
   |
   | question text / content
   |
   + --- Answer area ---
```

(continues on next page)

(continued from previous page)

```
  |
  | - [x] correct answer content
  | - [ ] answer content
  |
  |   optional feedback for an incorrect answer
  |
  + --------------------
```

The compact format is limited to a single list paragraph for an answer because all following paragraphs are assumed to be feedback. The first 'word' in a list *must* be *[x]* or *[ ]*.

Note

Alternate indicators

[x], *[X]*, and *[+]* are all synonyms for a correct answer.

[ ] and *[-]* are both synonyms for a wrong answer.

For complex answers and feedback, use nested feedback lists:

```
.. tb-choice::

   question text

   - answer content that can contain multiple blocks

     - feedback for an incorrect answer

   - answer content that can contain multiple blocks

     + feedback for a correct answer
```

3.2.2 Options

class
: `String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

name
: `String`. Optional. Sphinx reference name for this choice block. See *Common options* for details.

force-multiple
: `Boolean`. Optional. If present, HTML uses checkboxes even when only one answer is correct. This can keep the input type from revealing whether the question has one correct answer or more than one correct answer.

random
: `Boolean`. Optional. If present, HTML randomizes the answer order when the page loads. Text and PDF-oriented builders keep the authored order.

3.2.3 Sphinx configuration options

tb_choice_force_multiple
: `Boolean`. Optional. Default: `False`. If `True`, all `tb-choice` directives use checkboxes, even when only one answer is correct. A directive can also opt in with `:force-multiple:`.

tb_choice_random
: `Boolean`. Optional. Default: `False`. If `True`, all `tb-choice` directives randomize answer order when the page loads. A directive can also opt in with `:random:`.

3.2.4 Accessibility behavior

HTML uses native radio buttons or checkboxes. A native button checks the selection. The result text uses a status region so assistive technology can announce the result after checking.

3.2.5 Fallback behavior

HTML without JavaScript renders the prompt, answers, and feedback in document order. Text and PDF-oriented builders render the prompt and choices without correctness markers. Single-select questions use open-circle choice markers. Multiple-select questions use open-square choice markers. Feedback is omitted from paper-oriented output so the printed document can ask the complete question without revealing the answer.

3.2.6 Examples

Example 1: One correct answer

Source

```
.. tb-choice::

   What does the following code print when ``x`` has been set to 187?

   .. code-block:: java

      if (x < 0)
      {
          System.out.println("x is negative");
      }
      else if (x == 0)
      {
          System.out.println("x is zero");
      }
      else
      {
          System.out.println("x is positive");
      }

   - [ ] x is negative

     This will only print if x has been set to a number less than
     zero. Has it?

   - [ ] x is zero

     This will only print if x has been set to 0. Has it?
```

(continues on next page)

(continued from previous page)

```
   - [x] x is positive

     The first condition is false and ``x`` is not equal to zero, so
     the else block executes.
```

Rendered

Question

What does the following code print when **x** has been set to 187?

```
if (x < 0)
{
    System.out.println("x is negative");
}
else if (x == 0)
{
    System.out.println("x is zero");
}
else
{
    System.out.println("x is positive");
}
```

- ◯ x is negative
- ◯ x is zero
- ◯ x is positive

Example 2: More than one correct answer

Source

```
.. tb-choice::
   :random:

   Select the prime numbers.

   - [x] 2

     ``2`` is prime.

   - [x] 3

     ``3`` is prime.

   - [ ] 4

     ``4`` is composite.
```

Rendered

Question

Select the prime numbers.

- ☐ 2
- ☐ 3
- ☐ 4

Example 3: Compact choices without feedback

Source

```
.. tb-choice::
   :force-multiple:

   Which color is found in the rainbow?

   - [ ] Black
   - [x] Green
   - [ ] White
   - [ ] Brown
   - [ ] Gray
```

Rendered

Question

Which color is found in the rainbow?

- ☐ Black
- ☐ Green
- ☐ White
- ☐ Brown
- ☐ Gray

Example 4: List markup in the question

Source

```
.. tb-choice::

   Review the following observations:

   - The variable ``total`` starts at ``0``.
   - The loop adds each item in the list.
   - The loop stops after the final item.

   What value does ``total`` contain after summing ``2``, ``4``,
   and ``6``?

   - [ ] 6
```

(continues on next page)

(continued from previous page)

```
    This only includes the final item.

  - [ ] 10

    This misses one of the values in the list.

  - [x] 12

    ``2 + 4 + 6`` gives the final total.
```

Rendered

Question

Review the following observations:

- The variable `total` starts at `0`.
- The loop adds each item in the list.
- The loop stops after the final item.

What value does `total` contain after summing 2, 4, and 6?

◯ 6

◯ 10

◯ 12

Example 5: Nested feedback with math

Source

```
.. tb-choice::

   The Pythagorean theorem is :math:`a^2 + b^2 = c^2`. Given:

   .. math::

      3^2 + 4^2 = c^2

   What is the value of ``c``?

   - :math:`5`

     + Correct. The display calculation is:

       .. math::

          9 + 16 = 25 = 5^2

   - :math:`7`

     - This adds the side lengths directly. The theorem squares
```

(continues on next page)

(continued from previous page)

```
      each leg first:

      .. math::

         3^2 + 4^2 \neq 3 + 4

  - :math:`25`

    - This is :math:`c^2`, not ``c``. Take the square root:

      .. math::

         c = \sqrt{25} = 5
```

Rendered

Question

The Pythagorean theorem is $a^2 + b^2 = c^2$. Given:

$$3^2 + 4^2 = c^2$$

What is the value of `c`?

- ◯ 5
- ◯ 7
- ◯ 25

3.3 tb-click

The `tb-click` directive creates a click-on-source question. Authors provide normal prompt content, exactly one literal source block, and one or more `tb-hit` or `tb-miss` regions.

Use `tb-hit` for correct clickable regions. Use `tb-miss` for incorrect clickable regions with feedback. Selectors are exact, case-sensitive matches against the source text.

3.3.1 Synopsis

The general format of the `tb-click` directive is:

```
.. tb-click::
   :optional parameter: value

   + --- Prompt area ---
   |
   | question text and optional Sphinx content
   |
   + --- Source area ---
   |
   | exactly one code-block or literal block
   |
   + --- Region area ---
```

(continues on next page)

(continued from previous page)

```
|
| .. tb-hit:: selector
|
|    feedback for a correct click
|
| .. tb-miss:: selector
|
|    feedback for an incorrect click
|
+ -------------------
```

3.3.2 Selector Forms

Bare selector
A bare selector is the same as `text:`. It selects the first exact text match.

`text:literal`
Selects the first exact text match.

`text:literal#n`
Selects the nth exact text match.

`line:literal`
Selects the whole line that contains the exact text.

`range:line:start-end`
Selects a 1-based, inclusive line and column range. For example, `range:3:11-12` selects columns 11 through 12 on line 3.

3.3.3 Options

class
`String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

name
`String`. Optional. Sphinx reference name for this click question. See *Common options* for details.

show-hints
`Boolean`. Optional. If present, clickable source regions start with hint styling visible. By default, clickable regions match surrounding source text until focus, selection state, or the user chooses to show hints.

3.3.4 Sphinx configuration options

tb_click_show_hints
`Boolean` or `"never"`. Optional. Default: `False`. If `True`, hints are initially shown and the hint button starts with `Hide Hints`. If `False`, hints are initially hidden and the hint button starts with `Show Hints`. If `"never"`, hints are not shown and the hint button is omitted.

3.3.5 Accessibility behavior

HTML renders each clickable source region as a native button. The selected region receives visible state, feedback is shown, and result text uses a status region so assistive technology can announce the result after a click. Clickable regions have neutral accessible labels before selection, so correctness is not revealed before the user answers.

3.3.6 Fallback behavior

HTML without JavaScript renders the prompt, source, and feedback in document order. Text and PDF-oriented builders render the prompt and source only. Feedback is omitted from paper-oriented output so the printed document can ask the complete question without revealing the answer.

3.3.7 Examples

Example 1: SQL operator

Source

```
.. tb-click::

   Click the comparison operator.

   .. code-block:: sql

      SELECT name
      FROM students
      WHERE age >= 18;

   .. tb-hit:: >=

      ``>=`` is the comparison operator.

   .. tb-miss:: line:SELECT name

      This line selects output columns.

   .. tb-miss:: line:FROM students

      This line names the table.
```

Rendered

Click question

Click the comparison operator.

```
SELECT name
FROM students
WHERE age >= 18;
```

Example 2: C++ token occurrence

Source

```
.. tb-click::

   Click the second use of ``x``.

   .. code-block:: cpp

      int x = 3;
```

(continues on next page)

(continued from previous page)

```
   x = x + 1;
   std::cout << x;

.. tb-hit:: text:x#2

   This is the second occurrence of ``x``.

.. tb-miss:: int

   ``int`` is the type, not the requested occurrence.
```

Rendered

Click question

Click the second use of `x`.

```
int x = 3;
x = x + 1;
std::cout « x;
```

Example 3: Poetry line

Source

```
.. tb-click::

   Click the line that names the season.

   .. code-block:: text

      The woods are still.
      Autumn gathers at the gate.
      A lantern waits beside the road.

   .. tb-hit:: line:Autumn gathers

      This line names the season.

   .. tb-miss:: line:The woods are still.

      This line describes the setting.
```

Rendered

Click question

Click the line that names the season.

```
The woods are still.
Autumn gathers at the gate.
A lantern waits beside the road.
```

3.4 tb-code

The `tb-code` directive creates a runnable source-code block. In HTML, readers can send code to a remote server and optionally edit the source before running it again.

3.4.1 Synopsis

The general format of the `tb-code` directive is:

```
.. tb-code:: language
   :optional parameter: value

   + --- Source code area ---
   |
   | one or more lines of source code
   |
   + ------------------------
```

The source code area is required. The language can be supplied as the directive argument or as the `language` option.

3.4.2 Options

caption
: `String`. Optional. Standard `code-block` caption displayed with the static code listing. A caption alone does not create a hyperlink target. Combine `caption` with `name` when a block should be referenced with `:numref:`.

class
: `String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

compileargs, linkargs, runargs, interpreterargs
: `String` or `list`. Optional. Arguments passed through the Jobe `parameters` object. Use Python-style list syntax when an argument contains punctuation or spaces, for example `:compileargs: ['-Wall', '-std=c++11']`. A simple shell-style string such as `:runargs: --verbose sample.txt` is also accepted. When `runargs` is present, HTML shows the value in an editable text input.

dedent
: `Integer`. Optional. Standard Sphinx `code-block` dedent option passed through to the static highlighted listing. See the Sphinx code-block documentation for more information.

edit-label, hide-edit-label
: `String`. Optional. Labels for the edit toggle button. `edit-label` is used when the editor is hidden, and `hide-edit-label` is used when the editor is visible.

emphasize-lines
: `String`. Optional. Standard Sphinx `code-block` emphasize-lines option passed through to the static highlighted listing. See the Sphinx code-block documentation for more information.

endpoint
: `String`. Optional. Jobe-compatible `runs` endpoint for this code block. Defaults to `tb_code_default_endpoint`.

files
: `String`. Optional. Attaches one or more `tb-file` artifacts to the run request. Use filenames from the `tb-file filename` option, separated by spaces, commas, or new lines.

```
:files: input.txt data/config.json
```

Text files appear in HTML as editable file textareas unless the `tb-file` was marked `readonly`. Binary files are attached as read-only base64 content.

files-endpoint
`String`. Optional. Jobe-compatible `files` endpoint used to upload support files before a run. Defaults to `tb_code_files_endpoint`.

force
`Boolean`. Optional. Standard Sphinx `code-block` force option passed through to the static highlighted listing. See the Sphinx code-block documentation for more information.

hidden
`Boolean`. Optional. Hides rendered output. Hidden blocks can still be named and included by another `tb-code` block.

include
`String`. Optional. Replaces `{{PLACEHOLDER}}` tokens in this block with literal source copied from a named `tb-code`, `code-block`, or `literalinclude` directive. Use one mapping per line:

```
:include:
   PUBLIC_MEMBERS: account-methods
   PRIVATE_MEMBERS: account-fields
```

If the placeholder appears on its own indented line, the inserted source uses the same indentation. Include names use normal Sphinx reference names, so they can refer to named code in another source file. Duplicate include fragment names stop the build.

language
`String`. Optional. Source language. This may also be supplied as the directive argument. Default is configured by `tb_code_default_language`.

lineno-start
`Integer`. Optional. Standard Sphinx `code-block` lineno-start option passed through to the static highlighted listing. See the Sphinx code-block documentation for more information.

linenos
`Boolean`. Optional. Standard Sphinx `code-block` linenos option passed through to the static highlighted listing. See the Sphinx code-block documentation for more information.

name
`String`. Optional. Sphinx reference name for this runnable code block. Use `name` by itself for explicit-title references such as :ref:`run this code <example-code>`. Use `name` with `caption` for numbered references such as :numref:`example-code`. See *Common options* for details.

readonly
`Boolean`. Optional. Hides the edit control in HTML output.

revision-label
`String`. Optional. Label for the source version slider.

run-after
`String`. Optional. One or more named code fragments appended to the current source only when code is sent to the execution service. The appended source is not shown in the static listing, editor, or source version history.

Use one name per line, or separate names with commas:

```
:run-after:
   test-main
   assertions
```

run-before
: `String`. Optional. One or more named code fragments prepended to the current source only when code is sent to the execution service. The prepended source is not shown in the static listing, editor, or source version history.

  Use one name per line, or separate names with commas:

```
:run-before:
   imports
   setup-fixture
```

run-label
: `String`. Optional. Label for the HTML run button.

show-tutor
: `Boolean`. Optional. If present, HTML adds a Python Tutor button for supported languages: C, C++, Python, Java, and JavaScript. The button opens a new window with a Python Tutor permalink containing the current execution source, including any `run-before` and `run-after` fragments.

stdin
: `String`. Optional. Standard input sent with the run request. When present, HTML shows this value in an editable text input.

3.4.3 Reference behavior

`tb-code` follows Sphinx `code-block` reference conventions. Use `caption` for the visible listing caption. Use `name` for a stable hyperlink target. Use both options when a code block should be referenced with `:numref:`:

```
See :numref:`hello-code`.

.. tb-code:: python
   :name: hello-code
   :caption: Hello example

   print("Hello")
```

If only `name` is set, use an explicit-title `:ref:`:

```
See :ref:`this runnable example <hello-code>`.

.. tb-code:: python
   :name: hello-code

   print("Hello")
```

3.4.4 Sphinx configuration options

tb_code_default_endpoint
: `String`. Default Jobe-compatible run endpoint. The project default is `https://delicate-frost-8843.fly.dev/jobe/index.php/restapi/runs/`.

tb_code_languages_endpoint
: `String`. Jobe-compatible language discovery endpoint. The project default is `https://delicate-frost-8843.fly.dev/jobe/index.php/restapi/languages`.

tb_code_files_endpoint
: `String`. Jobe-compatible file upload endpoint. The project default is `https://delicate-frost-8843.fly.dev/jobe/index.php/restapi/files/`.

tb_code_validate_language
: Boolean. If true, query the languages supported before running code. If discovery reports that the configured language is unsupported, show a warning. If discovery is unavailable, execution still proceeds.

tb_code_default_language
: String. Default source language. The project default is python3.

tb_code_language_map
: dict. Optional aliases from author-facing language names to Jobe language identifiers. Use this when authors prefer a name that differs from the Jobe server's language ID, or when you want tb-code to match language names used elsewhere in the book.

  For example, {"python": "python3", "c++": "cpp", "js": "nodejs"} lets authors write .. tb-code:: python, .. tb-code:: c++, or .. tb-code:: js while Jobe receives python3, cpp, or nodejs. If the author-facing name already matches the Jobe ID, such as java to java, no mapping is needed.

tb_code_language_defaults
: dict. Optional per-language Jobe parameter defaults. Keys may be author-facing language names such as c++ or Jobe language identifiers such as cpp. Values are dictionaries containing compileargs, linkargs, runargs, or interpreterargs lists. If both an author-facing name and its mapped Jobe ID have defaults, the author-facing name takes precedence.

tb_code_block_defaults
: dict. Optional defaults for standard Sphinx code-block options used by tb-code. This is useful for presentation settings that should apply to every runnable code block, such as line numbers or custom styling.

```
tb_code_block_defaults = {
    "linenos": True,
    "lineno-start": 1,
    "class": ["touchbook-code"],
}
```

  Directive options override matching values from tb_code_block_defaults. Avoid setting name in this dictionary because reference names should be unique per block.

tb_code_run_label
: String. Default label for the run button. The project default is Run.

tb_code_edit_label
: String. Default label for the edit button when the editor is hidden. The project default is Edit.

tb_code_hide_edit_label
: String. Default label for the edit button when the editor is visible. The project default is Hide editor.

tb_code_revision_label
: String. Default label for the source version slider. The project default is Source version.

3.4.5 Service contract

Code that cannot be compiled natively in a browser is sent to a Jobe server for processing.

tb-code sends a Jobe-compatible request shaped like this:

```
{
  "run_spec": {
    "language_id": "cpp",
    "sourcecode": "int main() { return 0; }",
    "input": "Alice",
    "file_list": [["tbfileabc123", "input.txt"]],
```

(continues on next page)

(continued from previous page)

```
    "parameters": {
      "compileargs": ["-Wall", "-std=c++11"],
      "linkargs": [],
      "runargs": [],
      "interpreterargs": []
    }
  }
}
```

The directive options `compileargs`, `linkargs`, `runargs`, and `interpreterargs` populate the corresponding keys in `parameters`. Values set on an individual directive override matching values from `tb_code_language_defaults`. Defaults for keys not supplied on the directive still apply.

If `stdin` or `runargs` are configured, HTML presents editable text inputs initialized from those values. The current input values are used when the reader presses Run. These runtime inputs are separate from the source version history.

When `files` is configured, each attached file is uploaded to the configured Jobe `files` endpoint before the run request is sent. The run request then uses Jobe's `file_list` field to map each uploaded file identifier to the filename visible inside the execution workspace.

When `run-before` or `run-after` is configured, those named fragments are combined with the current source only for the execution request. They are not shown in the static listing or editor. Treat these fragments as hidden from the page interface, not as private code; generated HTML can still be inspected by a reader.

When `show-tutor` is configured for C, C++, Python, Java, or JavaScript, HTML adds a Python Tutor button. The permalink uses the same execution source that the Run button sends to Jobe. Runtime standard input, run arguments, and attached files are not included in the Python Tutor URL.

3.4.6 Accessibility behavior

The no-JS HTML fallback is a normal Sphinx-highlighted code listing. HTML adds native `button` controls, a `textarea` editor, a source version slider, optional runtime text inputs, a polite status region, and a labeled output region.

The edit button is a toggle. When the editor is opened, its label changes from `tb_code_edit_label` to `tb_code_hide_edit_label` and `aria-expanded` is updated. The source version slider starts with the original source as version 1 and appears after the editable source creates a second version. Once more than one version exists, the slider remains visible when the editor is hidden so readers can select and run an older source version. Loading a version updates the textarea and the visible source listing.

3.4.7 Fallback behavior

PDF and text builders render the source code as static readable content.

3.4.8 Examples

Example 1: Basic Python

Source

```
.. tb-code:: python

   print("Hello, world")
```

Rendered

```
print("Hello, world")
```

Example 2: Python defaults

Python usually does not require compile or link arguments, but defaults can still be set in `conf.py`. The example below maps `python` to Jobe's `python3` language identifier and provides an interpreter argument default.

conf.py

```
tb_code_default_language = "python3"
tb_code_language_map = {
    "python": "python3",
}
tb_code_language_defaults = {
    "python3": {
        "interpreterargs": ["-B"],
    },
}
```

Source

```
.. tb-code:: python
   :caption: Hello from Python
   :interpreterargs: ['-B']

   print("Hello, world")
```

Rendered

Listing 1: Hello from Python

```
print("Hello, world")
```

Example 3: Include named code

Use `include` to assemble a runnable block from named source fragments. This is useful when an example needs to show part of a program separately and later run it in a larger context.

Source

```
.. tb-code:: cpp
   :name: account-methods
   :hidden:

   public:
     int balance() const;

.. code-block:: cpp
   :name: account-fields
```

(continues on next page)

(continued from previous page)

```
  private:
    int balance_;

.. tb-code:: cpp
   :name: code-ex3
   :caption: Account class assembled from named code
   :include:
      PUBLIC_MEMBERS: account-methods
      PRIVATE_MEMBERS: account-fields

   class account {
     {{PUBLIC_MEMBERS}}
     {{PRIVATE_MEMBERS}}
   };
```

Rendered

```
private:
  int balance_;
```

Listing 2: Account class assembled from named code

```
class account {
  public:
    int balance() const;
  private:
    int balance_;
};
```

Example 4: Execution-only test runner

Use `run-after` when code should be sent to the execution service without being shown in the static listing or editor. This is useful for test runners or support code that would distract from the code students should read.

Source

```
.. tb-code:: cpp
   :name: account-balance-tests
   :hidden:

   int main() {
     account sample;
     return sample.balance() == 100 ? 0 : 1;
   }

.. tb-code:: cpp
   :name: code-ex4
   :caption: Account implementation with hidden tests
   :run-after: account-balance-tests
   :show-tutor:
```

(continues on next page)

(continued from previous page)

```
  class account {
  public:
    int balance() const {
      return 100;
    }
  };
```

Rendered

Listing 3: Account implementation with hidden tests

```
class account {
public:
  int balance() const {
    return 100;
  }
};
```

Example 5: Python command-line arguments

Use `runargs` when a program expects command-line arguments. In HTML, the initial value appears in an editable text input before the program runs.

Source

```
.. tb-code:: python
   :name: code-ex5
   :caption: Python command-line arguments
   :runargs: Ada Lovelace

   import sys

   if len(sys.argv) < 3:
       print("Please provide a first and last name.")
   else:
       first = sys.argv[1]
       last = sys.argv[2]
       print(f"Hello, {first} {last}!")
```

Rendered

Listing 4: Python command-line arguments

```
import sys

if len(sys.argv) < 3:
    print("Please provide a first and last name.")
else:
    first = sys.argv[1]
    last = sys.argv[2]
    print(f"Hello, {first} {last}!")
```

Example 6: Java

Java examples often need JVM limits. These can be configured once in `conf.py` and overridden on a specific `tb-code` block when needed.

conf.py

```
tb_code_language_defaults = {
    "java": {
        "interpreterargs": ["-Xrs", "-Xss8m", "-Xmx128m"],
    },
}
```

Source

```
.. tb-code:: java
   :name: code-ex6
   :caption: Fahrenheit to Celsius
   :emphasize-lines: 9
   :interpreterargs: ['-Xrs', '-Xss8m', '-Xmx128m']
   :stdin: 100

   import java.util.Scanner;

   public class TempConv {
       public static void main(String[] args) {
            Double fahr;
            Double cel;
            Scanner in;

            in = new Scanner(System.in);
            System.out.println("Enter the temperature in F: ");
            fahr = in.nextDouble();

            cel = (fahr - 32) * 5.0/9.0;
            System.out.println(fahr + " degrees F is: " + cel + " C");
       }

   }
```

Rendered

Listing 5: Fahrenheit to Celsius

```
1  import java.util.Scanner;
2
3  public class TempConv {
4      public static void main(String[] args) {
5           Double fahr;
6           Double cel;
7           Scanner in;
8
9           in = new Scanner(System.in);
```

(continues on next page)

(continued from previous page)

```
        System.out.println("Enter the temperature in F: ");
        fahr = in.nextDouble();

        cel = (fahr - 32) * 5.0/9.0;
        System.out.println(fahr + " degrees F is: " + cel + " C");
    }

}
```

Example 7: C++

For C++, use `compileargs` for compiler flags and `linkargs` for linker flags. The language map lets authors write `c++` while sending Jobe the `cpp` language identifier.

conf.py

```
tb_code_language_map = {
    "c++": "cpp",
    "cpp": "cpp",
}
tb_code_language_defaults = {
    "cpp": {
        "compileargs": ["-Wall", "-Wextra", "-pedantic", "-std=c++11"],
    },
}
```

Source

```
.. tb-code:: c++
   :name: code-ex7
   :caption: Hello from C++
   :compileargs: ['-Wall', '-Wextra', '-pedantic', '-std=c++11']
   :runargs: --repeat=3
   :stdin: Alice

   // A simple test for C++11 compiler
   #include <cstdlib>
   #include <iostream>
   #include <string>

   int main(int argc, char* argv[]) {
     int test[] = { 1, 2, 3, 5, 8 };
     for (auto i: test) {
       std::cout << "i is " << i << '\n';
     }

     int repeat = 1;
     for (int i = 1; i < argc; ++i) {
       std::string arg = argv[i];
       if (arg.find("--repeat=") == 0) {
         repeat = std::atoi(arg.substr(9).c_str());
```

(continues on next page)

(continued from previous page)

```
        if (repeat < 1) {
          repeat = 1;
        }
      }
    }

    std::string name;
    std::cin >> name;
    for (int i = 0; i < repeat; ++i) {
      std::cout << "Hello, " << name << "!\n";
    }
    return 0;
  }
```

Rendered

Listing 6: Hello from C++

```
// A simple test for C++11 compiler
#include <cstdlib>
#include <iostream>
#include <string>

int main(int argc, char* argv[]) {
  int test[] = { 1, 2, 3, 5, 8 };
  for (auto i: test) {
    std::cout << "i is " << i << '\n';
  }

  int repeat = 1;
  for (int i = 1; i < argc; ++i) {
    std::string arg = argv[i];
    if (arg.find("--repeat=") == 0) {
      repeat = std::atoi(arg.substr(9).c_str());
      if (repeat < 1) {
        repeat = 1;
      }
    }
  }

  std::string name;
  std::cin >> name;
  for (int i = 0; i < repeat; ++i) {
    std::cout << "Hello, " << name << "!\n";
  }
  return 0;
}
```

Example 8: GNU Octave

Octave is a powerful Scientific Programming Language designed to be largely compatible with Matlab. Although Octave does support built-in 2D/3D plotting and visualization tools those tools are not available through the touchbook interface. Output is limited to text.

Source

```
.. tb-code:: octave
   :name: code-ex8
   :caption: Solve Linear Algebra Equations with Octave

   # Define matrices
   A = [2, 1; 1, 3];
   B = [5, 6; 4, 7];

   # Solve matrix equation X = A\B
   X = A \ B
```

Rendered

Listing 7: Solve Linear Algebra Equations with Octave

```
# Define matrices
A = [2, 1; 1, 3];
B = [5, 6; 4, 7];

# Solve matrix equation X = A\B
X = A \ B
```

3.5 tb-file

The `tb-file` directive describes a simulated local file. A file can be shown in the document, hidden for later execution use, or attached to a `tb-code` execution request.

3.5.1 Synopsis

The general format of the `tb-file` directive is:

```
.. tb-file::
   :filename: required-filename
   :optional parameter: value

   + --- File content area ---
   |
   | one or more lines of file content
   |
   + -------------------------
```

The `filename` option is required. Inline content or a referenced source file is required.

The directive can also read content from a source file in the Sphinx project:

```
.. tb-file:: examples/input.txt
   :filename: input.txt
```

3.5.2 Options

filename
`String`. Required. The simulated local filename. This is the filename that code should expect when the file is attached to an execution environment.

Filenames may contain only letters, numbers, dots, underscores, hyphens, and forward slashes and must be relative paths. Empty path segments, `.`, `..`, absolute paths, spaces, and other punctuation are rejected.

caption
`String`. Optional. Caption displayed with the static file listing.

class
`String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

editable
`Boolean`. Optional. Text files are editable by default. This option can explicitly request editing if a project-level default later changes.

encoding
`String`. Optional. Encoding used when reading a referenced text file. Default is `utf-8`.

hidden
`Boolean`. Optional. If present, register the file but do not render it in output.

name
`String`. Optional. Sphinx reference name for this file artifact. See *Common options* for details.

readonly
`Boolean`. Optional. If present, do not add editing controls for text files in HTML. Binary files are never editable.

3.5.3 Sphinx configuration options

No directive-specific configuration options exist.

3.5.4 Accessibility behavior

Visible text files render their caption, or their filename when no caption is set, and their content as readable preformatted text before JavaScript runs. When editing is available, HTML adds a native button and textarea with programmatic labels.

Image files render with the simulated filename as alternate text. Authors should choose filenames that communicate the image's instructional role when using image files this way.

3.5.5 Fallback behavior

Text builders render visible text files as labeled file listings. PDF-oriented builders render visible text files as LaTeX listings. The listing caption uses `caption` when set and otherwise uses `filename`. Image files render as labeled image-file references. Hidden files are not rendered, but remain available in the Sphinx environment registry for later execution integration.

Referenced binary files are registered and marked read-only. HTML can display image formats supported by browsers. Text and PDF-oriented builders render a labeled file reference for binary files rather than embedding incompatible data. Authors should use builder-specific `only` sections when a binary file format is suitable for one output format but not another, such as a PostScript file for LaTeX-oriented output.

3.5.6 Examples

Example 1: Inline text file

Source

```
.. tb-file::
   :filename: input.txt
   :caption: Example input file

   Hello file!
```

Rendered

Listing 8: Example input file

```
Hello file!
```

Example 2: Hidden text file

Source

```
.. tb-file::
   :filename: data/input.txt
   :hidden:

   100
```

Rendered

The file is registered but not displayed.

Example 3: Reading a simulated file from C++

This example uses text from Lewis Carroll's *Jabberwocky*. When Run is pressed, the current contents of `poem_text` are sent with the execution request. If the file is edited in the attached-file editor, that edited content is sent instead of the original content.

Source

```
.. tb-file::
   :filename: poem_text

   Beware the Jabberwock, my son!
     The jaws that bite, the claws that catch!

.. tb-code:: c++
   :caption: Read a simulated local file
   :files: poem_text

   #include <fstream>
   #include <iostream>

   int main () {
```

(continues on next page)

(continued from previous page)

```
    std::ifstream is("poem_text");
    char c;
    while (is.get(c)) {
      std::cout << c;
    }
    is.close();
    return 0;
  }
```

Rendered

Listing 9: poem_text

```
Beware the Jabberwock, my son!
  The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
  The frumious Bandersnatch!"

He took his vorpal sword in hand:
  Long time the manxome foe he sought --
So rested he by the Tumtum tree,
  And stood awhile in thought.

And, as in uffish thought he stood,
  The Jabberwock, with eyes of flame,
Came whiffling through the tulgey wood,
  And burbled as it came!

One, two! One, two! And through and through
  The vorpal blade went snicker-snack!
He left it dead, and with its head
  He went galumphing back.

And, has thou slain the Jabberwock?
  Come to my arms, my beamish boy!
O frabjous day! Callooh! Callay!'
  He chortled in his joy.
```

– Lewis Carroll's *Jabberwocky*.

Listing 10: Read a simulated local file

```
#include <fstream>
#include <iostream>

int main () {
  std::ifstream is("poem_text");
  char c;
  while (is.get(c)) {
    std::cout << c;
  }
  is.close();
```

(continues on next page)

(continued from previous page)

```
  return 0;
}
```

3.6 tb-formula

The `tb-formula` directive creates a calculated numeric question. Authors define variable ranges, use variable placeholders in the question, and provide one formula that computes the expected answer from the generated values.

3.6.1 Synopsis

The general format of the `tb-formula` directive is:

```
.. tb-formula::
   :variables: x=min..max, y=min..max
   :optional parameter: value

   + --- Question area ---
   |
   | question text with {{x}} and {{y}} placeholders
   |
   + --- Answer formula ---

   .. answer-formula:: javascript

      formula using the variables
```

3.6.2 Options

variables
: `String`. Required. Comma-separated or semicolon-separated variable ranges. Each range uses `name=min..max` syntax. If both endpoints are integers, HTML generates integer values. Decimal endpoints generate decimal values.

class
: `String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

name
: `String`. Optional. Sphinx reference name for this calculated formula question. See *Common options* for details.

tolerance
: `Number`. Optional. Accepted absolute difference from a numeric formula result. The default is `0`. If the formula returns a range, `tolerance` is not used.

3.6.3 Sphinx configuration options

tb_formula_default_endpoint
: `String`. Optional. URL for remote formula execution when `answer-formula` uses a language other than `javascript` or `js`. The default is the same JOBE runs endpoint used by `tb-code`.

tb_formula_language_defaults
: `Dictionary`. Optional. Default JOBE parameters for remote formula languages. Supported keys are `compileargs`, `linkargs`, `runargs`, and `interpreterargs`.

3.6.4 Answer formula

The nested `answer-formula` block accepts an optional language argument. When omitted, the language is `javascript`.

JavaScript formulas run in the browser and can refer to each variable by name. Other languages are sent to a remote server execution endpoint. `tb-formula` sends the generated variables as JSON text on standard input. Results should print to standard output.

Remote formula blocks can set JOBE parameters with `:compileargs:`, `:linkargs:`, `:runargs:`, and `:interpreterargs:` options. These options accept either a shell-style string or a Python-style list of strings.

A formula can return:

- a number, such as `4 * y + 3 * x`
- a range of acceptable values as either:
  - a two-item range as a JSON array, such as `[359, 361]`
  - a JSON object with `min` and `max` keys

3.6.5 Accessibility behavior

HTML shows the generated values as text and uses a text input with decimal keyboard hints for the answer. The Check answer and New values controls are native buttons. Result text uses a status region so assistive technology can announce feedback.

3.6.6 Fallback behavior

HTML without JavaScript shows the question with placeholder blanks and an answer input. Text and PDF-oriented builders render the question with one deterministic value for each generated variable and a blank answer area. The static value is the midpoint of the configured range, rounded for integer ranges. Static output does not include the formula.

3.6.7 Examples

Example 1: Glasses of water

Source

```
.. tb-formula::
   :variables: x=4..8, y=10..20

   If a small glass can hold {{x}} ounces of water, and a large
   glass can hold {{y}} ounces of water, what's the total number of
   ounces in 4 large and 3 small glasses of water?

   .. answer-formula:: javascript

      4 * y + 3 * x
```

Rendered

Calculated formula

If a small glass can hold 6 ounces of water, and a large glass can hold 15 ounces of water, what's the total number of ounces in 4 large and 3 small glasses of water?

Example 2: Tolerance

Source

```
.. tb-formula::
   :variables: radius=2.0..5.0
   :tolerance: 0.05

   A circle has radius {{radius}}. What is its area?

   .. answer-formula:: javascript

      Math.PI * radius * radius
```

Rendered

Calculated formula

A circle has radius 3.5. What is its area?

Example 3: Fixed answer range

Source

```
.. tb-formula::
   :variables: width=10..20

   A board is {{width}} centimeters wide. Estimate its width to the
   nearest 5 centimeters.

   .. answer-formula:: javascript

      [width - 2.5, width + 2.5]
```

Rendered

Calculated formula

A board is 15 centimeters wide. Estimate its width to the nearest 5 centimeters.

Example 4: Data-dependent range

Source

```
.. tb-formula::
   :variables: distance=80..120, time=8..12

   A cart travels {{distance}} meters in {{time}} seconds. What is
   its speed in meters per second? Answers within 5 percent are
   accepted.

   .. answer-formula:: javascript
```

(continues on next page)

(continued from previous page)

```
    ({ min: (distance / time) * 0.95, max: (distance / time) * 1.05 })
```

Rendered

Calculated formula

A cart travels 100 meters in 10 seconds. What is its speed in meters per second? Answers within 5 percent are accepted.

Example 5: Remote formula program

Source

```
.. tb-formula::
   :variables: x=1..10, y=1..10

   What is {{x}} plus {{y}}?

   .. answer-formula:: python3

      import json
      import sys

      values = json.load(sys.stdin)
      print(values["x"] + values["y"])
```

Rendered

Calculated formula

What is 6 plus 6?

Example 6: Minimal C++ formula program

Source

```
.. tb-formula::
   :variables: n=2..9

   What is {{n}} squared?

   .. answer-formula:: cpp
      :compileargs: ['-std=c++11']

      #include <cctype>
      #include <iostream>
      #include <iterator>
      #include <string>

      int main() {
```

(continues on next page)

(continued from previous page)

```
    std::string input(
      (std::istreambuf_iterator<char>(std::cin)),
      std::istreambuf_iterator<char>()
    );
    std::string digits;
    for (char c : input) {
      if (std::isdigit(static_cast<unsigned char>(c))) {
        digits += c;
      }
    }
    int n = std::stoi(digits);
    std::cout << n * n << '\n';
    return 0;
  }
```

Rendered

Calculated formula

What is 6 squared?

3.7 tb-group

The `tb-group` directive is a container that splits related content into selectable tabs in HTML output.

3.7.1 Synopsis

The general format of the `tb-group` directive is:

```
.. tb-group::
   :optional parameter: value

   + --- Tab area ---
   |
   | .. tb-tab:: required tab label
   |
   |    one or more lines of tab content
   |
   | .. tb-tab:: required tab label
   |
   |    one or more lines of tab content
   |
   + ----------------
```

The `tb-group` directive must contain at least one immediate `tb-tab` directive and may contain only `tb-tab`s.

Content placed as an immediate child of `tb-group` that is not inside `tb-tab` is ignored by the tab interface.

There is no hard limit on the maximum number of tabs allowed. On narrow pages, the tabs wrap to fit the available width. Too many tabs can impair usability, so use judgment when grouping content.

3.7.2 Options

tab label
: `String`. Required for `tb-tab`. Creates a new tab and labels it with the provided string. The label may contain spaces.

  Any valid Sphinx markup can reside within a tab.

class
: `String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

name
: `String`. Optional. Sphinx reference name for this group or tab. See *Common options* for details.

3.7.3 Sphinx configuration options

No directive-specific configuration options exist.

3.7.4 Accessibility behavior

The no-JS HTML fallback shows each tab as a labeled content block. HTML creates an ARIA `tablist` with native `button` controls and keyboard support for Arrow Left, Arrow Right, Home, and End.

3.7.5 Fallback behavior

PDF and text builders render each tab as labeled static content.

3.7.6 Examples

Example 1: Basic tab group

Source

```
.. tb-group::

   .. tb-tab:: First tab

      This is the first tab.

   .. tb-tab:: Second tab

      This is the second tab.
```

Rendered

First tab

This is the first tab.

Second tab

This is the second tab.

Example 2: Sphinx markup inside tabs

Source

```
.. tb-group::
   :name: tab-ex2

   .. tb-tab:: Python
      :name: tab-ex2-python

      .. code-block:: python

         print("Hello from Python")

   .. tb-tab:: C++
      :name: tab-ex2-cpp

      .. code-block:: cpp

         #include <iostream>

         int main() {
             std::cout << "Hello from C++\n";
         }
```

Rendered

Python

```
print("Hello from Python")
```

C++

```
#include <iostream>

int main() {
    std::cout << "Hello from C++\n";
}
```

3.8 tb-match

The `tb-match` directive creates a matching question. Authors provide normal prompt content followed by one definition list. Each definition-list term is a source item. Each definition is the matching target.

HTML renders each source item beside a dropdown list of definitions. Students choose one definition for each source item, then use `Check Me` to evaluate the matches. Authors can add distractor definitions that appear as dropdown choices but do not match any source item.

3.8.1 Synopsis

The general format of the `tb-match` directive is:

```
.. tb-match::
   :optional parameter: value

   + --- Prompt area ---
   |
   | question text and optional Sphinx content
   |
   + --- Pair area ---
   |
   | source item
   |    matching target
   |
   | another source item
   |    another matching target
   |
   + -----------------
```

3.8.2 Options

distractors
: `String`. Optional. Additional definitions that do not match any source item. Separate multiple distractors with semicolons or continuation lines. Dropdown choices are sorted by visible definition text, so distractors do not always appear after the matching definitions.

class
: `String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

name
: `String`. Optional. Sphinx reference name for this matching question. See *Common options* for details.

3.8.3 Accessibility behavior

HTML uses native select controls so keyboard and assistive technology behavior comes from the browser. The result text uses a status region so assistive technology can announce the result after checking.

3.8.4 Fallback behavior

HTML without JavaScript renders source and target content in the page. Text builders render the prompt, a source list, and a target list. PDF-oriented builders render the prompt and a two-column matching table. Sources use upper-case letter labels. Targets include blank answer lines where readers can write the matching source letter.

3.8.5 Examples

Example 1: Terms and meanings

Source

```
.. tb-match::

   Match each term with its meaning.

   compiler
      Translates source code into executable code.
```

(continues on next page)

(continued from previous page)

```
  interpreter
     Executes source code directly.

  linker
     Combines object files into a program.
```

Rendered

Matching question

Match each term with its meaning.

A. compiler

B. interpreter

C. linker

___ Combines object files into a program.

___ Executes source code directly.

___ Translates source code into executable code.

Example 2: SQL clauses

Source

```
.. tb-match::
   :distractors:
      Sorts rows after filtering
      Groups rows by value

   Match each SQL clause with its purpose.

   ``SELECT``
      Chooses output columns.

   ``FROM``
      Names the source table.

   ``WHERE``
      Filters rows before they appear in the result.
```

Rendered

Matching question

Match each SQL clause with its purpose.

A. FROM

B. SELECT

C. WHERE

___ Chooses output columns.

___ Filters rows before they appear in the result.

___ Groups rows by value

___ Names the source table.

___ Sorts rows after filtering

Example 3: Simple Code

Source

```
.. tb-match::

   Match the function declaration to an example of its function call.

   int times_two(double x, string y)
      times_two(4.5,"hello");

   int times_two(string x, double y)
      times_two("hello", 10);

   int times_two(string x, string y)
      times_two("hello", "there");

   int times_two(int x, int y)
      times_two(4,7);
```

Rendered

Matching question

Match the function declaration to an example of its function call.

A. int times_two(double x, string y) ___ times_two(4,7);

B. int times_two(int x, int y) ___ times_two(4.5,”hello”);

C. int times_two(string x, double y) ___ times_two(“hello”, 10);

D. int times_two(string x, string y) ___ times_two(“hello”, “there”);

3.9 tb-micro-parsons

The `tb-micro-parsons` directive creates a token-level Parsons problem. Authors write tokens in the correct order. HTML shuffles the tokens and lets students move tokens from a source row into an answer row before checking the answer.

3.9.1 Synopsis

The general format of the `tb-micro-parsons` directive is:

```
.. tb-micro-parsons::
   :optional parameter: value

   + --- Optional prompt area ---
   |
   | question text and optional Sphinx content
   |
   + --- Token list in correct order ---

   - first-token
   - second-token
   - third-token
```

3.9.2 Options

class
`String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

name
`String`. Optional. Sphinx reference name for this micro-Parsons problem. See *Common options* for details.

distractor
`String`. Optional. Additional tokens that do not belong in the final answer. Tokens can be separated with semicolons or written one per continuation line.

3.9.3 Accessibility behavior

HTML renders each token as a native button. Activating a token in the source row moves it to the end of the answer row. Activating a token in the answer row moves it back to the source row. Keyboard focus remains on the token after it moves. Result text uses a status region so assistive technology can announce feedback after checking.

Distractors are omitted by leaving them in the source row.

3.9.4 Fallback behavior

HTML without JavaScript renders the prompt, a deterministic token order, and an empty answer row. Text builders render the prompt, shuffled tokens, and a blank answer line. PDF-oriented builders render the shuffled tokens in a literal block with a blank answer line. Static output does not support interactive checking.

3.9.5 Examples

Example 1: Assignment statement

Source

```
.. tb-micro-parsons::

   Arrange the tokens to create a valid assignment statement.

   - int
   - x
   - =
   - 42
   - ;
```

Rendered

Micro Parsons problem

Arrange the tokens to create a valid assignment statement.

Tokens

```
42 ; = int x
```

Answer: _____

Example 2: Distractor tokens

Source

```
.. tb-micro-parsons::
   :distractor: float; ==

   Arrange the tokens to create a valid assignment statement.

   - int
   - x
   - =
   - 42
   - ;
```

Rendered

Micro Parsons problem

Arrange the tokens to create a valid assignment statement.

Tokens

```
42 ; = == float int x
```

Answer: _____

Example 3: SQL clause

Source

```
.. tb-micro-parsons::
   :distractor:
      ORDER
      GROUP

   Arrange the tokens to create a SQL ``WHERE`` condition.

   - WHERE
   - score
   - >=
   - 90
```

Rendered

Micro Parsons problem

Arrange the tokens to create a SQL `WHERE` condition.

Tokens

```
90 >= GROUP ORDER score WHERE
```

Answer: _____

3.10 tb-order

The `tb-order` directive creates an ordering question. Authors can include optional prompt content before one bullet list. Authors write list items in the correct order. HTML presents the items in a shuffled order and lets students move them with native Up and Down buttons before checking the answer.

3.10.1 Synopsis

The general format of the `tb-order` directive is:

```
.. tb-order::
   :optional parameter: value

   + --- Optional prompt area ---
   |
   | question text and optional Sphinx content
   |
   + --- Ordered item list ---

   - first item in the correct order
   - second item in the correct order
   - third item in the correct order
```

3.10.2 Options

class
: `String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

name
: `String`. Optional. Sphinx reference name for this ordering question. See *Common options* for details.

3.10.3 Accessibility behavior

HTML uses native buttons for each reorder action. Each item has an Up button and a Down button. Buttons are disabled when an item cannot move farther in that direction. Result text uses a status region so assistive technology can announce feedback after checking.

3.10.4 Fallback behavior

HTML without JavaScript renders the prompt and a deterministic item order with movement buttons. Text and PDF-oriented builders render the prompt and items as an ordering question. Static output does not support interactive checking.

3.10.5 Examples

Example 1: Daily sequence

Source

```
.. tb-order::

   Put these events in order.

   - Wake up
```

(continues on next page)

(continued from previous page)

```
   - Eat breakfast
   - Go to class
```

Rendered

Ordering question

Put these events in order.

Items

- Eat breakfast
- Go to class
- Wake up

Example 2: C++ compilation stages

Source

```
.. tb-order::

   - Write source code
   - Preprocess source files
   - Compile translation units
   - Link object files
   - Run the executable
```

Rendered

Ordering question

Items

- Compile translation units
- Link object files
- Preprocess source files
- Run the executable
- Write source code

3.11 tb-parsons

The `tb-parsons` directive creates a Parsons problem. Authors write the fragments in the correct order. HTML shuffles the fragments and lets students move, indent, outdent, use, or skip each fragment before checking the answer.

3.11.1 Synopsis

The general format of the `tb-parsons` directive is:

```
.. tb-parsons::
   :optional parameter: value

```

(continues on next page)

(continued from previous page)

```
+ --- Optional prompt area ---
|
| question text and optional Sphinx content
|
+ --- Source block in correct order ---

.. code-block:: language

   first fragment
   {{group}}
       grouped fragment line one
       grouped fragment line two
   {{endgroup}}
   {{distractor}}
       optional distractor fragment
```

3.11.2 Options

class
: `String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

name
: `String`. Optional. Sphinx reference name for this Parsons problem. See *Common options* for details.

no-indent
: `Flag`. Optional. If present, indentation does not affect checking. Students still see indent controls, but only fragment order and skipped distractors are graded.

3.11.3 Authoring fragments

If the source block contains no markers, each non-blank source line becomes one fragment.

Use `{{group}}` and `{{endgroup}}` to make multiple source lines one fragment. Use this when a line and its body should move together. After `{{endgroup}}`, following non-blank lines become single-line fragments until another marker appears.

Use `{{distractor}}` to mark the following fragment as a distractor. HTML shows distractors with the other fragments. Students must mark distractors as skipped before checking their answer.

A `{{distractor}}` can precede a single line or a group:

```
{{distractor}}
single-line distractor

{{distractor}}
{{group}}
multi-line distractor
{{endgroup}}
```

3.11.4 Accessibility behavior

HTML uses native buttons for movement, indentation, and use/skip state. Each Use/Skip button keeps `aria-pressed` synchronized with its state and includes visible text. Result text uses a status region so assistive technology can announce feedback after checking.

3.11.5 Fallback behavior

HTML without JavaScript renders the prompt and a deterministic fragment order with controls. Text builders render the prompt and shuffled fragments as a Parsons problem. PDF-oriented builders render the shuffled fragments in a literal block. Static output does not support interactive checking.

3.11.6 Examples

Example 1: Function maximum

Source

```
.. tb-parsons::

   Construct a function that returns the max value from a list.

   .. code-block:: python

      def findmax(alist):
      {{group}}
          if len(alist) == 0:
              return None
      {{endgroup}}

          curmax = alist[0]
          for item in alist:
              if item > curmax:
                  curmax = item
          return curmax
```

Rendered

Parsons problem

Construct a function that returns the max value from a list.

Code fragments

```
curmax = alist[0]

curmax = item

def findmax(alist):

for item in alist:

if item > curmax:

if len(alist) == 0:
    return None

return curmax
```

Example 2: Distractor fragment

Source

```
.. tb-parsons::

   Construct a loop that prints each item in ``values``.

   .. code-block:: python

      for value in values:
      {{group}}
          print(value)
      {{endgroup}}
      {{distractor}}
          return value
```

Rendered

Parsons problem

Construct a loop that prints each item in `values`.

Code fragments

```
for value in values:

print(value)

return value
```

Example 3: Ignore indentation

Source

```
.. tb-parsons::
   :no-indent:

   Put the SQL clauses in order.

   .. code-block:: sql

      SELECT name
      FROM students
      WHERE active = 1
      ORDER BY name
```

Rendered

Parsons problem

Put the SQL clauses in order.

Code fragments

```
FROM students

ORDER BY name

SELECT name

WHERE active = 1
```

3.12 tb-reveal

The `tb-reveal` directive hides content until a reader asks to show it.

3.12.1 Synopsis

The general format of the `tb-reveal` directive is:

```
.. tb-reveal::
   :optional parameter: value

   + --- Content area ---
   |
   | one or more lines of initially hidden content
   | which can include any Touchbook or Sphinx supported directives.
   |
   + --------------------
```

The content area is required.

3.12.2 Options

class
: `String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

hidelabel
: `String`. Optional. Label for the hide or close button. Default is `Hide`.

modal
: `Boolean`. Optional. If included, the revealed content is presented in a modal dialog. The default behavior reveals content inline.

modal-titlebar
: `String`. Optional. Text displayed in the modal dialog titlebar and used as the dialog's accessible label. Default is `Message from the author`.

name
: `String`. Optional. Sphinx reference name for this reveal block. See *Common options* for details.

showlabel
: `String`. Optional. Label for the show button. Default is `Show`.

3.12.3 Sphinx configuration options

No directive-specific configuration options exist.

3.12.4 Accessibility behavior

The no-JS HTML fallback uses native `details` and `summary`. Inline HTML uses a native `button` and synchronizes `aria-expanded`. Modal content uses a native `dialog` element when available and includes an accessible dialog label.

3.12.5 Fallback behavior

PDF and text builders render the content as labeled static content.

3.12.6 Examples

Example 1: Basic reveal

Source

```
.. tb-reveal::

   This content starts out hidden.

   - *Any* valid `Sphinx markup <http://www.sphinx-doc.org>`__ can be
     included.
   - Hidden content can be shown by using the Show button.
   - When shown, a Hide button appears at the end of the hidden
     content.
```

Rendered

Show

This content starts out hidden.

- *Any* valid Sphinx markup can be included.
- Hidden content can be shown by using the Show button.
- When shown, a Hide button appears at the end of the hidden content.

Example 2: Custom button labels

Source

```
.. tb-reveal::
   :name: re-ex2
   :showlabel: Reveal Content
   :hidelabel: Hide Content

   The reveal block can contain other directives. This example uses a
   standard Sphinx code block:

   .. code-block:: python
```

(continues on next page)

(continued from previous page)

```
      print("Hello, world")

   and a ``tb-code`` block:

   .. tb-code:: python

      print("Hello, world")
```

Rendered

Reveal Content

The reveal block can contain other directives. This example uses a standard Sphinx code block:

```
print("Hello, world")
```

and a `tb-code` block:

```
print("Hello, world")
```

Example 3: Modal reveal

Source

```
Given the following C++ statements:

.. code-block:: cpp

   int  val = 0;
   int& ir  = val;
   auto x   = ir;

What type is x?

.. tb-reveal::
   :name: reveal-ex3
   :modal:
   :modal-titlebar: Understanding auto type deduction

   If you said, ``int``, excellent job!

   ``ir`` is a reference to ``val``,
   which makes ``ir`` just another name for ``val``.
   ``auto x = ir;`` is exactly the same as if we had written
   ``auto x = val;`` here.
```

Rendered

Given the following C++ statements:

```
int  val = 0;
int& ir  = val;
```

(continues on next page)

(continued from previous page)

```
auto x   = ir;
```

What type is x?

Understanding auto type deduction

If you said, `int`, excellent job!

`ir` is a reference to `val`, which makes `ir` just another name for `val`. `auto x = ir;` is exactly the same as if we had written `auto x = val;` here.

3.13 tb-video

The `tb-video` directive adds a video placeholder that can open a player in the page or in a separate window. The source URL can point to local static video files or to common provider pages from YouTube, Vimeo, Canvas LMS, and Odysee. YouTube opens in a separate window because provider iframe behavior depends on page origin and video settings. Any content in the body becomes optional notes for the video.

3.13.1 Synopsis

The general format of the `tb-video` directive is:

```
.. tb-video:: video-url
   :optional parameter: value

   + --- Notes area ---
   |
   | optional notes that explain the video or point to a
   | specific time in the recording.
   |
   + -------------------
```

The directive argument is required and provides the video URL or local file path.

3.13.2 Options

class
: `String` or `List`. Optional. A CSS class to add to the directive. See *Common options* for details.

height
: `String`. Optional. Height used for the player and placeholder frame. If omitted, the frame height is computed from the available width and the default aspect ratio. Authors can pass standard CSS-style values such as `640`, `640px`, `50%`, `12em`, `20rem`, and similar length or percentage forms. Docutils documents these values in the Common Option Value Types section.

name
: `String`. Optional. Sphinx reference name for this video block. See *Common options* for details.

thumbnail
: `String`. Optional. Image used in the placeholder frame. Can be a filename or URL. YouTube thumbnails are derived automatically. Use this option when another provider or local video should show a specific preview image.

width
`String`. Optional. Width used for the player and placeholder frame. If omitted, the frame uses the full available width.

window
`Boolean`. Optional. If present, HTML opens the video in a separate window instead of showing an inline player in the page. YouTube uses this behavior by default. Inline player support can be sensitive to browser type so consider forcing this setting if needed.

3.13.3 Sphinx configuration options

`tb_video_default_width`
`String`. Default width value passed to HTML configuration when `:width:` is omitted. The frame still uses the full available width unless `:width:` is set on the directive.

`tb_video_default_height`
`String`. Default height value passed to HTML configuration when `:height:` is omitted. The frame still computes height from its aspect ratio unless `:height:` is set on the directive.

3.13.4 Accessibility behavior

HTML renders a placeholder frame with a real link for no-JS access. The HTML uses native buttons and an iframe or video element with a programmatic label. For inline playback, clicking either the placeholder or the Play button loads the player in the page. The Play button then becomes a Pause button. For native video, Pause stops playback and leaves the controls visible. For iframe providers, Pause unloads the iframe and returns to the placeholder. Some providers may still require a click on their own player controls before playback begins.

3.13.5 Fallback behavior

HTML without JavaScript keeps the placeholder frame visible and the link operable. Text and PDF-oriented builders render the complete video URL and any notes from the content area. The notes stay part of the document flow so they remain available in non-HTML output.

3.13.6 Examples

Example 1: Local Ogg Vorbis video with inline playback

Ogg Theora/Vorbis playback depends on browser codec support. If the browser cannot play the file, HTML shows a message with a direct file link.

Source

```
.. tb-video:: _static/wilms-tumor-ct-scan.ogv
```

Rendered

Video

Video URL: _static/wilms-tumor-ct-scan.ogv

Example 2: Local WebM video with inline playback

Source

```
.. tb-video:: _static/Baby_Chick_Hatching.webm
   :width: 270
   :height: 480
   :thumbnail: _static/Baby_Chick_Hatching.jpg

   `Two Drs Homestead, CC BY 3.0 <https://creativecommons.org/licenses/by/3.0>`__,
   via Wikimedia Commons.
```

Rendered

Video

Video URL: _static/Baby_Chick_Hatching.webm

Notes.

Two Drs Homestead, CC BY 3.0, via Wikimedia Commons.

Example 3: Vimeo with inline playback

Source

```
.. tb-video:: https://vimeo.com/486845755

   Jump to the end of the lecture for the review question.
```

Rendered

Video

Video URL: https://vimeo.com/486845755

Notes.

Jump to the end of the lecture for the review question.

Example 4: Vimeo in a separate window

Source

```
.. tb-video:: https://vimeo.com/486845755
   :window:

   Jump to the end of the lecture for the review question.
```

Rendered

Video

Video URL: https://vimeo.com/486845755

Notes.

Jump to the end of the lecture for the review question.

Example 5: YouTube opens externally

Source

```
.. tb-video:: https://www.youtube.com/watch?v=aqz-KE-bpKQ&t=75s

   Start at 1:15 for the worked example.
```

Rendered

Video

Video URL: https://www.youtube.com/watch?v=aqz-KE-bpKQ&t=75s

Notes.

Start at 1:15 for the worked example.

CHAPTER

FOUR

ACCESSIBILITY AND KEYBOARD USE

Touchbook directives use native HTML controls wherever possible. Buttons, inputs, text areas, selects, links, and dialogs keep their browser keyboard behavior and programmatic labels. This helps assistive technology and lets each browser expose controls in the way users expect.

4.1 Keyboard Basics

Most Touchbook controls follow standard browser behavior:

- `Tab` moves forward through focusable controls.
- `Shift+Tab` moves backward through focusable controls.
- `Enter` activates links and most buttons.
- `Space` activates buttons, checkboxes, radio buttons, and similar controls.
- Arrow keys may move within composite widgets, such as tab groups.

4.2 Tab Groups

The `tb-group` directive renders an ARIA tab interface. Only the selected tab is in the normal `Tab` order. This is intentional and follows the common roving-tabindex pattern for tabs.

When focus is on a tab:

- `ArrowRight` moves to the next tab.
- `ArrowLeft` moves to the previous tab.
- `Home` moves to the first tab.
- `End` moves to the last tab.
- `Tab` moves into the selected tab panel when that panel contains a focusable control.

For example, if a `tb-group` has `Source` and `Rendered` tabs, `Tab` does not normally move from `Source` to `Rendered`. Use the arrow keys to select `Rendered`. Then use `Tab` to enter the rendered content.

4.3 Platform Settings

Keyboard navigation can depend on operating system and browser settings. If `Tab` skips buttons or other native controls, check the platform settings before assuming the page is broken.

macOS
: In System Settings, open **Keyboard** and enable full keyboard navigation. Safari also has a browser-specific setting in **Advanced** named **Press Tab to highlight each item on a webpage**.

When full keyboard access is disabled, macOS browsers may require `Option+Tab` or `Alt+Tab` to move to buttons and other controls.

Windows

Windows browsers usually include buttons in the normal `Tab` order. If navigation seems incomplete, check browser accessibility settings and any installed keyboard or assistive-technology utilities. In Microsoft Edge and Chrome, also check whether caret browsing or extension settings are changing keyboard behavior.

Linux

Linux behavior depends on the desktop environment, browser, and assistive technology stack. GNOME, KDE, Firefox, Chrome, and Chromium usually include native buttons in the normal `Tab` order. If they do not, check desktop keyboard accessibility settings, browser settings, and screen-reader or extension configuration.

4.4 Testing Guidance

When testing Touchbook content for keyboard accessibility:

- Test with full keyboard navigation enabled.
- Confirm that every visible control can receive focus.
- Confirm that visible focus is easy to see.
- Activate buttons with `Space` and `Enter`.
- Test `tb-group` tabs with arrow keys, not only with `Tab`.
- Test the static fallback output when building text or PDF formats.

Touchbook should provide accessible controls and predictable focus behavior. Operating system and browser settings can still change how users move through native controls.

Download the PDF version.