

typsphinx

typsphinx contributors

0.4.0

1 Getting Started

Contents

2	typsphinx Documentation	6
2.1	Key Features	6
2.2	Quick Links	6
3	Installation	6
3.1	Requirements	6
3.2	Installing from PyPI	6
3.3	Installing from Source	6
3.4	Development Installation	7
3.5	Verifying Installation	7
3.6	Next Steps	7
4	Quick Start	7
4.1	Basic Setup	7
4.2	Your First PDF	7
4.3	Configuration Options	8
4.4	What's Next?	8
5	User Guide	8
6	Configuration	8
6.1	Basic Configuration	9
6.2	Template Configuration	9
6.3	Math Rendering	10
6.4	Code Highlighting	10
6.5	Author Information	10
6.6	Paper Size and Format	10
6.7	Complete Example	11
6.8	See Also	11
7	Builders	12
7.1	Overview	12
7.2	typst Builder	12
7.3	typstopdf Builder	12
7.4	Configuration	13
7.5	Choosing a Builder	13
7.6	Common Workflow	13
7.7	See Also	14
8	Templates	14
8.1	Template System Overview	14
8.2	Default Template	14
8.3	Configuration-Based Templates	14
8.4	Custom Template Files	15
8.5	Template Parameters	16
8.6	Wrapping External Packages	16
8.7	Examples	17
8.8	Best Practices	18
8.9	Debugging Templates	18
8.10	See Also	19
8.11	Overview	19

8.12	Main Topics	19
9	Configuration	19
9.1	Basic Configuration	19
9.2	Template Configuration	20
9.3	Math Rendering	20
9.4	Code Highlighting	20
9.5	Author Information	21
9.6	Paper Size and Format	21
9.7	Complete Example	21
9.8	See Also	22
10	Builders	22
10.1	Overview	22
10.2	typst Builder	22
10.3	typstpdf Builder	23
10.4	Configuration	23
10.5	Choosing a Builder	24
10.6	Common Workflow	24
10.7	See Also	24
11	Templates	24
11.1	Template System Overview	25
11.2	Default Template	25
11.3	Configuration-Based Templates	25
11.4	Custom Template Files	25
11.5	Template Parameters	26
11.6	Wrapping External Packages	27
11.7	Examples	28
11.8	Best Practices	29
11.9	Debugging Templates	29
11.10	See Also	29
12	Examples	29
13	Basic Examples	29
13.1	Minimal Configuration	29
13.2	Adding Math	30
13.3	Code Blocks	30
13.4	Tables	31
13.5	Images	31
13.6	Cross-References	32
13.7	Lists and Admonitions	32
13.8	Complete Basic Example	33
13.9	Next Steps	34
14	Advanced Examples	34
14.1	Using Typst Universe Packages	34
14.2	Custom Template Wrapping	35
14.3	Multi-Document Projects	36
14.4	Custom Styling	37
14.5	Conditional Content	38
14.6	Bibliographies	38
14.7	Advanced Math	39
14.8	CI/CD Integration	39

14.9	Performance Optimization	40
14.10	See Also	41
14.11	Overview	41
15	Basic Examples	41
15.1	Minimal Configuration	41
15.2	Adding Math	42
15.3	Code Blocks	42
15.4	Tables	42
15.5	Images	43
15.6	Cross-References	43
15.7	Lists and Admonitions	44
15.8	Complete Basic Example	44
15.9	Next Steps	46
16	Advanced Examples	46
16.1	Using Typst Universe Packages	46
16.2	Custom Template Wrapping	47
16.3	Multi-Document Projects	48
16.4	Custom Styling	48
16.5	Conditional Content	49
16.6	Bibliographies	50
16.7	Advanced Math	50
16.8	CI/CD Integration	51
16.9	Performance Optimization	52
16.10	See Also	52
17	API Reference	52
17.1	Builders	53
17.2	Writer and Translator	58
17.3	Template Engine	86
17.4	Configuration	89
18	Indices and Tables	89
19	Contributing	89
19.1	Development Setup	90
19.2	Running Tests	90
19.3	Code Quality	90
19.4	Development Workflow	91
19.5	Coding Guidelines	92
19.6	Project Structure	93
19.7	Reporting Issues	93
19.8	Feature Requests	94
19.9	Community	94
19.10	Code of Conduct	94
19.11	License	94
20	Changelog	94
20.1	Version 0.4.0 (Current)	94
20.2	Version 0.3.0	94
20.3	Version 0.2.2	95
20.4	Version 0.2.1	95
20.5	Version 0.2.0	95
20.6	Version 0.1.0	95

20.7	Migration Guides	95
20.8	Development Status	96
20.9	Deprecation Policy	96
20.10	Upcoming Features	96
20.11	Versioning	96
20.12	Release Process	96
20.13	See Also	97
21	Indices and tables	97

2 typsphinx Documentation

typsphinx is a Sphinx extension that integrates the Sphinx documentation generator with the Typst typesetting system.

It combines Sphinx's powerful documentation generation capabilities with Typst's modern typesetting features to create high-quality technical documents.

2.1 Key Features

- **Sphinx to Typst Conversion:** Convert reStructuredText/Markdown to Typst format
- **Dual Builder Integration:**
 - `typst` builder: Generate Typst markup files
 - `typstpdf` builder: Generate PDF directly (no external Typst CLI required)
- **Self-Contained PDF Generation:** Self-contained PDF generation via `typst-py`
- **Customizable Output:** Customize Typst templates and styles
- **Cross-References:** Reproduce Sphinx cross-references, indexes, and table of contents in Typst
- **Code Highlighting:** Syntax highlighting with `codly` package
- **Math Support:** LaTeX math via `mitex` or native Typst math
- **Figure Management:** Embed and reference images, tables, and figures

2.2 Quick Links

- **GitHub Repository:** <https://github.com/YuSabo90002/typsphinx>
- **PyPI Package:** <https://pypi.org/project/typsphinx/>
- **Issue Tracker:** <https://github.com/YuSabo90002/typsphinx/issues>

3 Installation

3.1 Requirements

- Python 3.9 or later
- Sphinx 5.0 or later

3.2 Installing from PyPI

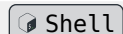
The easiest way to install `typsphinx` is using `pip`:

```
pip install typsphinx
```



Or using `uv` (recommended for faster installation):

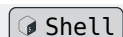
```
uv pip install typsphinx
```



3.3 Installing from Source

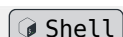
If you want to install from the latest source code:

```
git clone https://github.com/YuSabo90002/typsphinx.git
cd typsphinx
pip install -e .
```



Or with `uv`:

```
git clone https://github.com/YuSabo90002/typsphinx.git
cd typsphinx
uv pip install -e .
```



3.4 Development Installation

For development with all dependencies:

```
git clone https://github.com/YuSabo90002/typsphinx.git
cd typsphinx
uv sync --extra dev
```

Shell

This installs:

- All runtime dependencies
- Testing tools (pytest, pytest-cov)
- Code quality tools (black, ruff, mypy)
- Documentation tools

3.5 Verifying Installation

To verify that typsphinx is installed correctly:

```
python -c "import typsphinx; print(typsphinx.__version__)"
```

Shell

You should see the version number printed.

3.6 Next Steps

Continue to [Quick Start](#) to learn how to use typsphinx in your Sphinx project.

4 Quick Start

This guide will help you get started with typsphinx in just a few minutes.

4.1 Basic Setup

1. **Install typsphinx** (if you haven't already):

```
pip install typsphinx
```

Shell

2. **Add to your Sphinx project** Add typsphinx to the extensions list in your `conf.py`:

```
extensions = [
    "typsphinx",
]
```

Python

i Info

text("Thanks to entry points, adding to ") raw("extensions") text(" is optional.nThe builders are automatically discovered.")

3. **Build Typst output:**

```
# Generate Typst markup
sphinx-build -b typst source/ build/typst

# Or generate PDF directly
sphinx-build -b typstpdf source/ build/pdf
```

Shell

4.2 Your First PDF

Here's a minimal example to generate your first PDF:

1. Create a simple index.rst:

```
Welcome to My Documentation rst
=====

This is a sample document.

Features
-----

- Easy to use
- Beautiful PDFs
- Fast compilation
```

2. Build the PDF:

```
sphinx-build -b typstpdf source/ build/pdf Shell
```

3. Find your PDF in build/pdf/index.pdf!

4.3 Configuration Options

You can customize the output by adding options to conf.py:

```
# Project information Python
project = "My Project"
author = "Your Name"
release = "1.0.0"

# Typst configuration
typst_documents = [
    ("index", "myproject", project, author, "typst"),
]

# Use mitem for LaTeX math
typst_use_mitem = True

# Custom template (optional)
typst_template = "_templates/custom.typ"
```

4.4 What's Next?

- Learn about Configuration options
- Explore Builders (typst vs typstpdf)
- Customize with Templates
- See Examples for more examples

5 User Guide

This section provides comprehensive documentation on using typsphinx.

6 Configuration

This page documents all configuration options available for typsphinx.

6.1 Basic Configuration

Add these settings to your `conf.py` file:

6.1.1 Project Information

```
project = "My Project"
copyright = "2025, Author Name"
author = "Author Name"
release = "1.0.0"
```


 Python

These are standard Sphinx settings that `typsphinx` uses for document metadata.

6.1.2 Typst Documents

Define which documents to build:

```
typst_documents = [
    ("index", "output", "Title", "Author", "typst"),
]
```

 Python

Each tuple contains:

1. **Source file** (without `.rst` extension)
2. **Output filename** (without extension)
3. **Document title**
4. **Author**
5. **Document class** (usually “typst”)


6.2 Template Configuration

6.2.1 Template Function

Specify the Typst template function:

```
# Simple string format
typst_template_function = "project"


# Dictionary format with parameters
typst_template_function = {
    "name": "ieee",
    "params": {
        "abstract": "This paper presents...",
        "index-terms": ["AI", "ML"],
    }
}
```

 Python

6.2.2 Custom Template File

Use a custom Typst template file:

```
typst_template = "_templates/custom.typ"
```

 Python

The template file should define a project function (or the function specified in `typst_template_function`).

6.2.3 Typst Package

Use external Typst packages from Typst Universe:

```
typst_package = "@preview/charged-ieee:0.1.4"
```

 Python

6.3 Math Rendering

6.3.1 mitex Support

Enable LaTeX math rendering with mitex:

```
typst_use_mitex = True # Default
```

 Python

When enabled, LaTeX math expressions are converted to Typst using the mitex package.


When disabled, math is passed directly as Typst math syntax.

6.4 Code Highlighting

6.4.1 Codly Configuration


Enable code highlighting with codly:

```
typst_use_codly = True # Default
```

 Python

Customize line numbering:


```
typst_code_line_numbers = True # Show line numbers
```

 Python

6.5 Author Information

6.5.1 Simple Format


```
typst_author = ("John Doe", "Jane Smith")
```

 Python

6.5.2 Detailed Format


Include detailed author information:

```
typst_authors = {  
  "John Doe": {  
    "department": "Computer Science",  
    "organization": "MIT",  
    "email": "john@mit.edu"  
  },  
  "Jane Smith": {  
    "department": "Engineering",  
    "organization": "Stanford",  
    "email": "jane@stanford.edu"  
  }  
}
```

 Python

6.6 Paper Size and Format

```
typst_papersize = "a4" # Default: "a4"  
# Options: "a4", "us-letter", "a5", etc.
```

 Python

```
typst_fontsize = "11pt" # Default: "11pt"
```

6.7 Complete Example

Here's a complete `conf.py` example:

```
# Project information
project = "My Documentation"
copyright = "2025, My Name"
author = "My Name"
release = "1.0.0"

# General configuration
extensions = ["typsphinx"]

# Typst documents
typst_documents = [
    ("index", "mydoc", project, author, "typst"),
]

# Template configuration
typst_package = "@preview/charged-ieee:0.1.4"
typst_template_function = {
    "name": "ieee",
    "params": {
        "abstract": "This document demonstrates...",
        "index-terms": ["Documentation", "Typst"],
        "paper-size": "us-letter",
    }
}

# Author details
typst_authors = {
    "My Name": {
        "department": "Engineering",
        "organization": "My Organization",
        "email": "me@example.com"
    }
}

# Math and code
typst_use_mitex = True
typst_use_codly = True
typst_code_line_numbers = True
```

6.8 See Also

- Builders - Understanding the `typst` and `typstpdf` builders
- Templates - Customizing Typst templates

- Advanced Examples - Advanced configuration examples

7 Builders

typsphinx provides two builders for different use cases.

7.1 Overview

Builder	Output	Use Case
typst	.typ files	Edit Typst markup manually, use external Typst CLI
typstpdf	.pdf files	Direct PDF generation, CI/CD pipelines

7.2 typst Builder

The typst builder generates Typst markup files (.typ).

7.2.1 Usage

```
sphinx-build -b typst source/ build/typst
```

Shell

7.2.2 Output

- Generates .typ files in the output directory
- One file per document defined in typst_documents
- Include files for multi-document projects

7.2.3 When to Use

- You want to edit the generated Typst markup
- You have a specific Typst CLI version
- You need fine control over compilation
- You want to learn Typst syntax

7.2.4 Manual Compilation

After generating .typ files, compile with Typst CLI:

```
# Install Typst CLI if needed
# https://github.com/typst/typst

# Compile to PDF
typst compile build/typst/index.typ output.pdf
```

Shell

7.3 typstpdf Builder

The typstpdf builder generates PDF files directly using typst-py.

7.3.1 Usage

```
sphinx-build -b typstpdf source/ build/pdf
```

Shell

7.3.2 Output

- Generates .pdf files directly
- No external tools required
- Uses typst-py Python bindings

7.3.3 When to Use

- You want PDF output immediately

- You're running in CI/CD without Typst CLI
- You want self-contained builds
- You don't need to edit Typst markup

7.3.4 Advantages


- **No external dependencies:** Everything runs in Python
- **Faster setup:** No need to install Typst CLI
- **Reproducible builds:** Same output across environments
- **CI/CD friendly:** Works in restricted environments

7.4 Configuration

Both builders share the same configuration options in `conf.py`.

7.4.1 Document Definitions

```
typst_documents = [
    # (source, output, title, author, class)
    ("index", "main", "My Document", "Author", "typst"),
    ("api", "api-ref", "API Reference", "Author", "typst"),
]
```

 Python

7.4.2 Builder-Specific Options

There are no builder-specific options currently. All `typst_*` configuration options apply to both builders.

7.5 Choosing a Builder

Use `typst` if:

- You want to customize the generated Typst code
- You need specific Typst CLI features
- You're learning Typst and want to see the markup

Use `typstpdf` if:


- You just want PDF output
- You're building in CI/CD
- You want the simplest workflow
- You don't need to edit Typst code

7.6 Common Workflow

7.6.1 Development

During development, use `typstpdf` for quick feedback:

```
sphinx-build -b typstpdf source/ build/pdf
open build/pdf/index.pdf
```


 Shell

7.6.2 Production

For production, you can use either builder:

```
# Option 1: Direct PDF (recommended)
sphinx-build -b typstpdf source/ build/pdf

# Option 2: Typst + manual compilation
```

 Shell

```
sphinx-build -b typst source/ build/typst
typst compile build/typst/index.typ output.pdf
```

7.6.3 CI/CD

In CI/CD, typstpdf is recommended for simplicity:

```
- name: Build Documentation PDF
  run: |
    pip install typstpdf
    sphinx-build -b typstpdf docs/source docs/build/pdf
```

YAML

7.7 See Also

- Configuration - Configuration options
- Templates - Customizing templates
- Basic Examples - Basic usage examples

8 Templates

Customize the appearance and structure of your Typst output using templates.

8.1 Template System Overview

typstpdf uses Typst templates to control document layout and styling.

There are three ways to customize templates:

1. **Default template:** Built-in template (no configuration needed)
2. **Configuration-based:** Use `typst_template_function` dict format
3. **Custom template file:** Provide your own `.typ` template

8.2 Default Template

The default template provides a clean, professional layout:

```
# No configuration needed - uses built-in template
typst_documents = [
  ("index", "output", "Title", "Author", "typst"),
]
```

Python

Features:

- Title page with project name and author
- Table of contents
- Section numbering
- Professional styling

8.3 Configuration-Based Templates

Use Typst Universe packages with configuration:

```
typst_package = "@preview/charged-ieee:0.1.4"

typst_template_function = {
  "name": "ieee",
  "params": {
    "abstract": "This paper presents...",
```

Python

```

    "index-terms": ["AI", "ML"],
    "paper-size": "us-letter",
  }
}

```

Advantages:

- No custom files needed
- Declarative configuration
- Easy to maintain

See Advanced Examples for complete examples.

8.4 Custom Template Files

For full control, create a custom template file.

8.4.1 Basic Structure

Create a file `_templates/custom.typ`:

```

#let project(
  title: "",
  authors: (),
  date: none,
  body
) = {
  // Title page
  align(center)[
    #text(size: 24pt, weight: "bold")[#title]
    #v(1em)
    #text(size: 14pt)[#authors.join(", ")]
    #if date != none {
      v(1em)
      text(size: 12pt)[#date]
    }
  ]

  pagebreak()

  // Table of contents
  outline(title: "Contents", indent: auto)

  pagebreak()


  // Document body
  body
}

```

8.4.2 Configuration

Reference your custom template in `conf.py`:

```
typst_template = "_templates/custom.typ"
```

 Python

8.5 Template Parameters

8.5.1 Standard Parameters


Your template function receives these parameters:

- title: Document title (from `typst_documents`)
- authors: Author(s) tuple or list
- date: Document date (auto-generated or custom)
- body: The main document content

8.5.2 Custom Parameters


Add custom parameters using `typst_template_function`:

```
typst_template_function = {  
    "name": "project", # Your template function name  
    "params": {  
        "subtitle": "A Technical Report",  
        "version": "1.0",  
        "confidential": True,  
    }  
}
```

 Python

Access in template:


```
#let project(  
    title: "",  
    subtitle: none,  
    version: none,  
    confidential: false,  
    body  
) = {  
    // Use custom parameters  
    if confidential {  
        text(fill: red)[CONFIDENTIAL]  
    }  
    // ...  
}
```

 Typst

8.6 Wrapping External Packages

You can wrap Typst Universe packages in custom templates:

```
#import "@preview/charged-ieee:0.1.4": ieee  
  
#let project(  
    title: "",  
    authors: (),  
    body  
) = {
```

 Typst


```
// Transform parameters
let ieee_authors = authors.map(name => (
  name: name,
  department: "Engineering",
  organization: "My Org",
))

// Apply IEEE template
show: ieee.with(
  title: title,
  authors: ieee_authors,
)

body
}
```

This approach gives you:

- Parameter transformation
- Custom preprocessing
- Multiple package integration


8.7 Examples

8.7.1 Minimal Template

```
#let project(title: "", body) = {
  set page(paper: "a4", margin: 2.5cm)
  set text(font: "New Computer Modern", size: 11pt)


  align(center)[#text(20pt, weight: "bold")[#title]]
  v(2em)

  body
}
```

 Typst

8.7.2 Academic Paper Template

```
#let project(
  title: "",
  authors: (),
  abstract: none,
  keywords: (),
  body
) = {
  // Two-column layout
  set page(
    paper: "us-letter",
    columns: 2,
    margin: (x: 2cm, y: 2.5cm),
```

 Typst

```

)

// Title and authors in single column
place(top + center, float: true)[
  #text(18pt, weight: "bold")[#title]
  #v(0.5em)
  #text(12pt)[#authors.join(", ")]
]

// Abstract box
if abstract != none {
  place(top + center, float: true, clearance: 3em)[
    #box(width: 100%, inset: 1em)[
      *Abstract:* #abstract
    ]
  ]
}

// Keywords
if keywords.len() > 0 {
  place(top + center, float: true, clearance: 6em)[
    *Keywords:* #keywords.join(", ")
  ]
}

v(8em)

// Two-column body
body
}

```

8.8 Best Practices


1. **Start simple:** Use the default template or configuration-based approach first
2. **Reuse packages:** Leverage Typst Universe packages when possible
3. **Test incrementally:** Build frequently to catch errors early
4. **Document parameters:** Comment your template parameters clearly
5. **Keep it maintainable:** Don't over-complicate templates

8.9 Debugging Templates

If you encounter errors:

1. **Check syntax:** Typst errors are reported in build output
2. **Test standalone:** Compile your template with test data
3. **Use typst builder:** Generate .typ files to inspect output
4. **Simplify:** Remove customizations until it works

```
# Generate .typ files for inspection
sphinx-build -b typst source/ build/typst
```

 Shell

```
# Check the generated template usage
cat build/typst/index.typ
```

8.10 See Also

- Configuration - Template configuration options
- Advanced Examples - Advanced template examples
- Typst Documentation - Official Typst docs
- Typst Universe - Template packages

8.11 Overview

typsphinx integrates Sphinx with Typst to provide high-quality PDF output without the complexity of LaTeX.

The extension provides two builders:

- **typst**: Generates Typst markup files (`.typ`)
- **typstpdf**: Generates PDF files directly using `typst-py`

8.12 Main Topics

Configuration

Learn about all configuration options available in `conf.py`

Builders

Understand the difference between `typst` and `typstpdf` builders

Templates

Customize output using Typst templates

9 Configuration


This page documents all configuration options available for `typsphinx`.

9.1 Basic Configuration

Add these settings to your `conf.py` file:

9.1.1 Project Information

```
project = "My Project"
copyright = "2025, Author Name"
author = "Author Name"
release = "1.0.0"
```


 Python

These are standard Sphinx settings that `typsphinx` uses for document metadata.

9.1.2 Typst Documents

Define which documents to build:

```
typst_documents = [
    ("index", "output", "Title", "Author", "typst"),
]
```

 Python

Each tuple contains:

1. **Source file** (without `.rst` extension)

2. **Output filename** (without extension)
3. **Document title**
4. **Author**
5. **Document class** (usually “typst”)


9.2 Template Configuration

9.2.1 Template Function

Specify the Typst template function:

```
# Simple string format
typst_template_function = "project"


# Dictionary format with parameters
typst_template_function = {
  "name": "ieee",
  "params": {
    "abstract": "This paper presents...",
    "index-terms": ["AI", "ML"],
  }
}
```

 Python

9.2.2 Custom Template File

Use a custom Typst template file:

```
typst_template = "_templates/custom.typ"
```


 Python

The template file should define a project function (or the function specified in `typst_template_function`).

9.2.3 Typst Package

Use external Typst packages from Typst Universe:

```
typst_package = "@preview/charged-ieee:0.1.4"
```


 Python

9.3 Math Rendering

9.3.1 mitex Support

Enable LaTeX math rendering with mitex:

```
typst_use_mitex = True # Default
```

 Python


When enabled, LaTeX math expressions are converted to Typst using the mitex package. When disabled, math is passed directly as Typst math syntax.

9.4 Code Highlighting

9.4.1 Codly Configuration


Enable code highlighting with codly:

```
typst_use_codly = True # Default
```

 Python

Customize line numbering:

```
typst_code_line_numbers = True # Show line numbers
```

 Python

9.5 Author Information

9.5.1 Simple Format


```
typst_author = ("John Doe", "Jane Smith")
```

 Python

9.5.2 Detailed Format


Include detailed author information:

```
typst_authors = {  
    "John Doe": {  
        "department": "Computer Science",  
        "organization": "MIT",  
        "email": "john@mit.edu"  
    },  
    "Jane Smith": {  
        "department": "Engineering",  
        "organization": "Stanford",  
        "email": "jane@stanford.edu"  
    }  
}
```

 Python

9.6 Paper Size and Format


```
typst_papersize = "a4" # Default: "a4"  
# Options: "a4", "us-letter", "a5", etc.  
  
typst_fontsize = "11pt" # Default: "11pt"
```

 Python

9.7 Complete Example

Here's a complete `conf.py` example:

```
# Project information  
project = "My Documentation"  
copyright = "2025, My Name"  
author = "My Name"  
release = "1.0.0"  
  
# General configuration  
extensions = ["typsphinx"]  
  
# Typst documents  
typst_documents = [  
    ("index", "mydoc", project, author, "typst"),  
]  
  
# Template configuration
```

 Python

```

typst_package = "@preview/charged-ieee:0.1.4"
typst_template_function = {
  "name": "ieee",
  "params": {
    "abstract": "This document demonstrates...",
    "index-terms": ["Documentation", "Typst"],
    "paper-size": "us-letter",
  }
}

# Author details
typst_authors = {
  "My Name": {
    "department": "Engineering",
    "organization": "My Organization",
    "email": "me@example.com"
  }
}

# Math and code
typst_use_mitex = True
typst_use_codly = True
typst_code_line_numbers = True

```

9.8 See Also

- Builders - Understanding the typst and typstpdf builders
- Templates - Customizing Typst templates
- Advanced Examples - Advanced configuration examples

10 Builders

typst provides two builders for different use cases.

10.1 Overview


Builder	Output	Use Case
typst	.typ files	Edit Typst markup manually, use external Typst CLI
typstpdf	.pdf files	Direct PDF generation, CI/CD pipelines

10.2 typst Builder

The typst builder generates Typst markup files (.typ).

10.2.1 Usage

```
sphinx-build -b typst source/ build/typst
```

 Shell

10.2.2 Output

- Generates .typ files in the output directory
- One file per document defined in typst_documents
- Include files for multi-document projects

10.2.3 When to Use

- You want to edit the generated Typst markup
- You have a specific Typst CLI version
- You need fine control over compilation
- You want to learn Typst syntax

10.2.4 Manual Compilation

After generating `.typ` files, compile with Typst CLI:

```
# Install Typst CLI if needed
# https://github.com/typst/typst

# Compile to PDF
typst compile build/typst/index.typ output.pdf
```

Shell

10.3 typstpdf Builder

The `typstpdf` builder generates PDF files directly using `typst-py`.

10.3.1 Usage

```
sphinx-build -b typstpdf source/ build/pdf
```

Shell

10.3.2 Output

- Generates `.pdf` files directly
- No external tools required
- Uses `typst-py` Python bindings

10.3.3 When to Use

- You want PDF output immediately
- You're running in CI/CD without Typst CLI
- You want self-contained builds
- You don't need to edit Typst markup

10.3.4 Advantages

- **No external dependencies:** Everything runs in Python
- **Faster setup:** No need to install Typst CLI
- **Reproducible builds:** Same output across environments
- **CI/CD friendly:** Works in restricted environments

10.4 Configuration

Both builders share the same configuration options in `conf.py`.

10.4.1 Document Definitions

```
typst_documents = [
    # (source, output, title, author, class)
    ("index", "main", "My Document", "Author", "typst"),
    ("api", "api-ref", "API Reference", "Author", "typst"),
]
```

Python

10.4.2 Builder-Specific Options

There are no builder-specific options currently. All `typst_*` configuration options apply to both builders.

10.5 Choosing a Builder

Use `typst` if:

- You want to customize the generated Typst code
- You need specific Typst CLI features
- You're learning Typst and want to see the markup

Use `typstpdf` if:

- You just want PDF output
- You're building in CI/CD
- You want the simplest workflow
- You don't need to edit Typst code

10.6 Common Workflow

10.6.1 Development

During development, use `typstpdf` for quick feedback:

```
sphinx-build -b typstpdf source/ build/pdf
open build/pdf/index.pdf
```

Shell

10.6.2 Production

For production, you can use either builder:

```
# Option 1: Direct PDF (recommended)
sphinx-build -b typstpdf source/ build/pdf

# Option 2: Typst + manual compilation
sphinx-build -b typst source/ build/typst
typst compile build/typst/index.typ output.pdf
```

Shell

10.6.3 CI/CD

In CI/CD, `typstpdf` is recommended for simplicity:

```
- name: Build Documentation PDF
  run: |
    pip install typsphinx
    sphinx-build -b typstpdf docs/source docs/build/pdf
```

YAML

10.7 See Also

- Configuration - Configuration options
- Templates - Customizing templates
- Basic Examples - Basic usage examples

11 Templates

Customize the appearance and structure of your Typst output using templates.

11.1 Template System Overview

typsphinx uses Typst templates to control document layout and styling.


There are three ways to customize templates:

1. **Default template:** Built-in template (no configuration needed)
2. **Configuration-based:** Use `typst_template_function` dict format
3. **Custom template file:** Provide your own `.typ` template

11.2 Default Template

The default template provides a clean, professional layout:

```
# No configuration needed - uses built-in template
typst_documents = [
    ("index", "output", "Title", "Author", "typst"),
]
```

 Python

Features:


- Title page with project name and author
- Table of contents
- Section numbering
- Professional styling

11.3 Configuration-Based Templates

Use Typst Universe packages with configuration:

```
typst_package = "@preview/charged-ieee:0.1.4"

typst_template_function = {
    "name": "ieee",
    "params": {
        "abstract": "This paper presents...",
        "index-terms": ["AI", "ML"],
        "paper-size": "us-letter",
    }
}
```

 Python

Advantages:

- No custom files needed
- Declarative configuration
- Easy to maintain

See Advanced Examples for complete examples.


11.4 Custom Template Files

For full control, create a custom template file.

11.4.1 Basic Structure

Create a file `_templates/custom.typ`:

```
#let project(
    title: "",
    authors: (),
```

 Typst

```

    date: none,
    body
) = {
    // Title page
    align(center)[
        #text(size: 24pt, weight: "bold")[#title]
        #v(1em)
        #text(size: 14pt)[#authors.join(", ")]
        #if date != none {
            v(1em)
            text(size: 12pt)[#date]
        }
    ]

    pagebreak()

    // Table of contents
    outline(title: "Contents", indent: auto)

    pagebreak()


    // Document body
    body
}

```

11.4.2 Configuration

Reference your custom template in `conf.py`:

```
typst_template = "_templates/custom.typ"
```

 Python

11.5 Template Parameters

11.5.1 Standard Parameters

Your template function receives these parameters:

- title: Document title (from `typst_documents`)
- authors: Author(s) tuple or list
- date: Document date (auto-generated or custom)
- body: The main document content


11.5.2 Custom Parameters

Add custom parameters using `typst_template_function`:

```

typst_template_function = {
    "name": "project", # Your template function name
    "params": {
        "subtitle": "A Technical Report",
        "version": "1.0",
        "confidential": True,
    }
}


```

 Python

```
}  
}
```

Access in template:


```
#let project(  
  title: "",  
  subtitle: none,  
  version: none,  
  confidential: false,  
  body  
) = {  
  // Use custom parameters  
  if confidential {  
    text(fill: red)[CONFIDENTIAL]  
  }  
  // ...  
}
```

 Typst

11.6 Wrapping External Packages

You can wrap Typst Universe packages in custom templates:

```
#import "@preview/charged-ieee:0.1.4": ieee  
  
#let project(  
  title: "",  
  authors: (),  
  body  
) = {  
  // Transform parameters  
  let ieee_authors = authors.map(name => (  
    name: name,  
    department: "Engineering",  
    organization: "My Org",  
  ))  
  
  // Apply IEEE template  
  show: ieee.with(  
    title: title,  
    authors: ieee_authors,  
  )  
  
  body  
}
```

 Typst

This approach gives you:

- Parameter transformation
- Custom preprocessing

- Multiple package integration


11.7 Examples

11.7.1 Minimal Template

```
#let project(title: "", body) = {
  set page(paper: "a4", margin: 2.5cm)
  set text(font: "New Computer Modern", size: 11pt)

  align(center)[#text(20pt, weight: "bold")[#title]]
  v(2em)

  body
}
```

 Typst


11.7.2 Academic Paper Template

```
#let project(
  title: "",
  authors: (),
  abstract: none,
  keywords: (),
  body
) = {
  // Two-column layout
  set page(
    paper: "us-letter",
    columns: 2,
    margin: (x: 2cm, y: 2.5cm),
  )

  // Title and authors in single column
  place(top + center, float: true)[
    #text(18pt, weight: "bold")[#title]
    #v(0.5em)
    #text(12pt)[#authors.join(", ")]
  ]

  // Abstract box
  if abstract != none {
    place(top + center, float: true, clearance: 3em)[
      #box(width: 100%, inset: 1em)[
        *Abstract:* #abstract
      ]
    ]
  }

  // Keywords
```

 Typst

```

if keywords.len() > 0 {
    place(top + center, float: true, clearance: 6em)[
        *Keywords:* #keywords.join(", ")
    ]
}

v(8em)

// Two-column body
body
}

```

11.8 Best Practices

1. **Start simple:** Use the default template or configuration-based approach first
2. **Reuse packages:** Leverage Typst Universe packages when possible
3. **Test incrementally:** Build frequently to catch errors early
4. **Document parameters:** Comment your template parameters clearly
5. **Keep it maintainable:** Don't over-complicate templates

11.9 Debugging Templates

If you encounter errors:

1. **Check syntax:** Typst errors are reported in build output
2. **Test standalone:** Compile your template with test data
3. **Use typst builder:** Generate .typ files to inspect output
4. **Simplify:** Remove customizations until it works

```

# Generate .typ files for inspection
sphinx-build -b typst source/ build/typst

# Check the generated template usage
cat build/typst/index.typ

```

Shell

11.10 See Also

- Configuration - Template configuration options
- Advanced Examples - Advanced template examples
- Typst Documentation - Official Typst docs
- Typst Universe - Template packages

12 Examples

This section provides practical examples of using typsphinx.

13 Basic Examples

Simple examples to get started with typsphinx.

13.1 Minimal Configuration

The simplest possible setup:

conf.py:

```
project = "My Project"
author = "My Name"
extensions = ["typsphinx"]

typst_documents = [
    ("index", "output", project, author, "typst"),
]
```

 Python

index.rst:

```
My Documentation
=====

Welcome to my documentation!


Introduction
-----

This is a simple example.
```

 rst

Build:

```
sphinx-build -b typstpdf source/ build/pdf
```

 Shell

13.2 Adding Math

Include mathematical expressions:

index.rst:

```
Math Examples
=====

Inline math: :math:`E = mc^2`

Display math:

.. math::

    \int_0^{\infty} e^{-x^2} dx = \frac{\sqrt{\pi}}{2}
```

 rst

The math is automatically rendered using mitem (LaTeX) or native Typst math.

13.3 Code Blocks

Add syntax-highlighted code:

index.rst:

```
Code Example
=====
```

 rst

Python code:

```
.. code-block:: python

def hello(name):
    print(f"Hello, {name}!")

hello("World")
```

Code is highlighted using the `codly` package.

13.4 Tables

Create tables:

index.rst:

Data Table

rst

=====

```
.. list-table:: Feature Comparison
   :header-rows: 1
```

- * - Feature
 - typsphinx
 - LaTeX
- * - Setup time
 - 5 minutes
 - 2 hours
- * - PDF quality
 - Excellent
 - Excellent
- * - Ease of use
 - Easy
 - Complex

13.5 Images

Include images:

index.rst:

Figures

rst

=====

```
.. figure:: _static/diagram.png
   :width: 80%
   :align: center
```

Figure 1: System Architecture

Make sure to create a `_static/` directory for your images.

13.6 Cross-References

Link to sections and documents:

index.rst:

```
Documentation Structure rst
=====

See :ref:`installation-section` for setup instructions.

.. _installation-section:

Installation
-----

Installation instructions here.
```

Another file (api.rst):

```
API Reference rst
=====

See :doc:`/index` for the main documentation.
```

13.7 Lists and Admonitions

index.rst:

```
Important Notes rst
=====

.. note::

    This is a note admonition.

.. warning::

    This is a warning admonition.

Bullet list:

- Item 1
- Item 2
- Item 3

Numbered list:

1. First
```


2. Second

3. Third

13.8 Complete Basic Example

Here's a complete minimal project:

Directory structure:

```
myproject/
├─ source/
│   ├─ conf.py
│   ├─ index.rst
│   └─ _static/
└─ build/
```

conf.py:

```
project = "My Project"
copyright = "2025, My Name"
author = "My Name"
release = "1.0"

extensions = ["typsphinx"]

typst_documents = [
    ("index", "myproject", project, author, "typst"),
]

typst_use_mitex = True
typst_use_codly = True
```

index.rst:

```
My Project Documentation
=====

.. toctree::
   :maxdepth: 2

   introduction
   usage
   api

Introduction
-----

This project does amazing things.
```

Features:

- Feature 1
- Feature 2
- Feature 3


Quick Example

```
.. code-block:: python
```

```
import myproject
result = myproject.do_something()
print(result)
```

Build and view:

```
sphinx-build -b typstpdf source/ build/pdf
open build/pdf/myproject.pdf
```

 Shell

13.9 Next Steps

- Explore Advanced Examples for more complex examples
- Read Configuration for all options
- Learn about Templates for customization

14 Advanced Examples

Advanced configurations and use cases for typstpdf.

14.1 Using Typst Universe Packages

14.1.1 charged-ieee Template

Use the charged-ieee package for IEEE-style papers:


conf.py:

```
project = "Machine Learning Applications"
author = "John Doe"

# Use IEEE package
typst_package = "@preview/charged-ieee:0.1.4"

# Configure template with parameters
ieee_abstract = """
This paper presents novel approaches to machine learning
applications in computer vision.
"""

ieee_keywords = ["Machine Learning", "Computer Vision", "AI"]
```

 Python

```

typst_template_function = {
  "name": "ieee",
  "params": {
    "abstract": ieee_abstract,
    "index-terms": ieee_keywords,
    "paper-size": "us-letter",
  }
}


# Detailed author information
typst_authors = {
  "John Doe": {
    "department": "Computer Science",
    "organization": "MIT",
    "email": "john@mit.edu"
  }
}

```

14.1.2 modern-cv Template

Create a CV/resume with modern-cv:

```
typst_package = "@preview/modern-cv:0.7.0"
```

 Python

```

typst_template_function = {
  "name": "modern-cv",
  "params": {
    "name": "John Doe",
    "job-title": "Software Engineer",
    "email": "john@example.com",
    "github": "johndoe",
  }
}

```

14.2 Custom Template Wrapping

Wrap external packages with custom logic:


`_templates/custom_ieee.typ`:

```

#import "@preview/charged-ieee:0.1.4": ieee

#let project(
  title: "",
  authors: (),
  date: none,
  body
) = {
  // Transform simple author tuples to IEEE format

```

 Typst

```

let ieee_authors = authors.map(name => (
  name: name,
  department: "Engineering",
  organization: "My Organization",
  location: "City, State",
  email: name.split(" ").at(0).lower() + "@example.com"
))

// Define abstract and keywords (could be parameters)
let ieee_abstract = [
  This document demonstrates custom template wrapping
  with automatic author transformation.
]

let ieee_keywords = (
  "Documentation",
  "Typst",
  "Automation"
)

// Apply IEEE template
show: ieee.with(
  title: title,
  authors: ieee_authors,
  abstract: ieee_abstract,
  index-terms: ieee_keywords,
  bibliography: "refs.bib",
)

body
}


```

conf.py:

```

typst_template = "_templates/custom_ieee.typ"
typst_package = "@preview/charged-ieee:0.1.4"

```

 Python

14.3 Multi-Document Projects


Build multiple related documents:

conf.py:

```

typst_documents = [
  # (source, output, title, author, class)
  ("index", "main", "Main Documentation", "Team", "typst"),
  ("api/index", "api-reference", "API Reference", "Team", "typst"),
  ("tutorial/index", "tutorial", "Tutorial", "Team", "typst"),
]

```

 Python

Each document is built separately with its own output file.

14.4 Custom Styling

Apply custom fonts and colors:

`_templates/styled.typ`:

```
#let project(
  title: "",
  primary-color: blue,
  body
) = {
  // Set custom font
  set text(
    font: "New Computer Modern",
    size: 11pt,
  )

  // Custom heading style
  show heading.where(level: 1): it => {
    set text(fill: primary-color, size: 20pt, weight: "bold")
    it
    v(0.5em)
  }

  show heading.where(level: 2): it => {
    set text(fill: primary-color.lighten(20%), size: 16pt)
    it
    v(0.3em)
  }

  // Title page
  align(center)[
    #text(size: 28pt, fill: primary-color, weight: "bold")[
      #title
    ]
  ]

  pagebreak()

  body
}
```

`conf.py`:

```
typst_template = "_templates/styled.typ"

typst_template_function = {
```

```

    "name": "project",
    "params": {
        "primary-color": "rgb(0, 102, 204)", # Custom blue
    }
}

```

14.5 Conditional Content

Use different templates for different documents:

conf.py:

```

# Default template for most documents
typst_template = "_templates/default.typ"

# Define multiple documents
typst_documents = [
    ("index", "main", "Main Docs", "Team", "typst"),
    ("paper", "research-paper", "Research Paper", "Authors", "typst"),
]

```

For document-specific templates, you can use Sphinx's per-file configuration or conditional logic in your template.

14.6 Bibliographies

Include bibliographies with BibTeX:

index.rst:

```

Research Paper
=====

According to Smith et al. [Smith2023]_, machine learning...

References
-----

.. [Smith2023] Smith, J. (2023). Machine Learning Advances.
    Journal of AI Research, 15(2), 123-145.

```

Custom template with bibliography:

```

#let project(title: "", body) = {
    set page(paper: "us-letter")

    text(20pt, weight: "bold")[#title]
    v(2em)

    body

    // Bibliography section

```

```

pagebreak()
heading(numbering: none)[References]
// Bibliography rendered from .bib file
}

```

14.7 Advanced Math

Complex mathematical expressions:

index.rst:

Advanced Mathematics

rst

=====

Matrix equation:

.. math::

$$\mathbf{A} \mathbf{x} = \mathbf{b}$$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$=$$

$$\begin{pmatrix} x^2 + 2x + 1 \\ (x + 1)^2 \end{pmatrix}$$

$$\begin{pmatrix} f(x) \\ g(x) \end{pmatrix}$$

$$\begin{pmatrix} f(x) \\ g(x) \end{pmatrix}$$

$$=$$

$$\begin{pmatrix} x^2 + 2x + 1 \\ (x + 1)^2 \end{pmatrix}$$

$$\begin{pmatrix} f(x) \\ g(x) \end{pmatrix}$$

$$\begin{pmatrix} f(x) \\ g(x) \end{pmatrix}$$

Aligned equations:

.. math::

$$\begin{aligned} f(x) &= x^2 + 2x + 1 \\ g(x) &= (x + 1)^2 \end{aligned}$$

$$f(x) = x^2 + 2x + 1$$

$$g(x) = (x + 1)^2$$

$$f(x) = x^2 + 2x + 1$$

14.8 CI/CD Integration

GitHub Actions workflow for documentation:

.github/workflows/docs.yml:

name: Documentation

YAML YAML

on:

```

push:
  branches: [main]
pull_request:
  branches: [main]

jobs:
  build-docs:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Python
        uses: actions/setup-python@v5
        with:
          python-version: "3.11"

      - name: Install dependencies
        run: |
          pip install -e .
          pip install sphinx furo sphinx-autodoc-typehints

      - name: Build HTML documentation
        run: |
          cd docs
          sphinx-build -b html source _build/html

      - name: Build PDF documentation
        run: |
          cd docs
          sphinx-build -b typstpdf source _build/pdf

      - name: Upload PDF artifact
        uses: actions/upload-artifact@v4
        with:
          name: documentation-pdf
          path: docs/_build/pdf/*.pdf

```

14.9 Performance Optimization

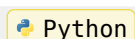
For large documentation projects:

conf.py:

```

# Parallel build
import multiprocessing
parallel_read_safe = True
parallel_write_safe = True

```




```
# Limit depth for faster builds
typst_documents = [
    ("index", "output", "Title", "Author", "typst"),
]

# Disable expensive features if not needed
typst_use_codly = True # Keep code highlighting
typst_code_line_numbers = False # Disable if not needed
```

14.10 See Also

- Configuration - All configuration options
- Templates - Template system details
- Typst Documentation - Official Typst reference
- Typst Universe - Package repository

14.11 Overview

The examples are organized by complexity:

Basic Examples

Simple examples to get started quickly

Advanced Examples

Advanced configurations and use cases

All examples are tested and ready to use in your projects.

15 Basic Examples

Simple examples to get started with typsphinx.


15.1 Minimal Configuration

The simplest possible setup:

conf.py:

```
project = "My Project"
author = "My Name"
extensions = ["typsphinx"]

typst_documents = [
    ("index", "output", project, author, "typst"),
]
```

 Python

index.rst:

```
My Documentation
=====

Welcome to my documentation!

Introduction
-----
```

 rst

```
This is a simple example.
```

Build:

```
sphinx-build -b typstpdf source/ build/pdf
```

Shell

15.2 Adding Math

Include mathematical expressions:

index.rst:

Math Examples

rst

=====

Inline math: `:math:`E = mc^2``

Display math:

`.. math::`

```
\int_0^{\infty} e^{-x^2} dx = \frac{\sqrt{\pi}}{2}
```

The math is automatically rendered using mitex (LaTeX) or native Typst math.

15.3 Code Blocks

Add syntax-highlighted code:

index.rst:

Code Example

rst

=====

Python code:

`.. code-block:: python`

```
def hello(name):  
    print(f"Hello, {name}!")
```

```
hello("World")
```

Code is highlighted using the codly package.

15.4 Tables

Create tables:

index.rst:

Data Table

rst

=====

```
.. list-table:: Feature Comparison
   :header-rows: 1
```

- | |
|-----------------|
| * - Feature |
| - typsphinx |
| - LaTeX |
| * - Setup time |
| - 5 minutes |
| - 2 hours |
| * - PDF quality |
| - Excellent |
| - Excellent |
| * - Ease of use |
| - Easy |
| - Complex |

15.5 Images

Include images:

index.rst:

```
Figures
```

rst

```
=====
```

```
.. figure:: _static/diagram.png
   :width: 80%
   :align: center
```

```
Figure 1: System Architecture
```

Make sure to create a `_static/` directory for your images.

15.6 Cross-References

Link to sections and documents:

index.rst:

```
Documentation Structure
=====
```

rst

```
See :ref:`installation-section` for setup instructions.
```

```
.. _installation-section:
```

```
Installation
```

```
-----
```

```
Installation instructions here.
```

Another file (api.rst):

```
API Reference
```

rst

```
=====
```

```
See :doc:`/index` for the main documentation.
```

15.7 Lists and Admonitions

index.rst:

```
Important Notes
```

rst

```
=====
```

```
.. note::
```

```
    This is a note admonition.
```

```
.. warning::
```

```
    This is a warning admonition.
```

```
Bullet list:
```

- Item 1
- Item 2
- Item 3

```
Numbered list:
```

1. First
2. Second
3. Third

15.8 Complete Basic Example

Here's a complete minimal project:

Directory structure:

```
myproject/
```

Text

```
├─ source/
```

```
│   └─ conf.py
```

```
│   └─ index.rst
```

```
│   └─ _static/
```

```
└─ build/
```

conf.py:

```
project = "My Project"
```

Python

```
copyright = "2025, My Name"
```

```

author = "My Name"
release = "1.0"

extensions = ["typsphinx"]

typst_documents = [
    ("index", "myproject", project, author, "typst"),
]

typst_use_mitex = True
typst_use_codly = True

```

index.rst:

```

My Project Documentation rst
=====

.. toctree::
   :maxdepth: 2

   introduction
   usage
   api

Introduction
-----

This project does amazing things.

Features:

- Feature 1
- Feature 2
- Feature 3

Quick Example
-----

.. code-block:: python

   import myproject
   result = myproject.do_something()
   print(result)

```

Build and view:

```
sphinx-build -b typstpdf source/ build/pdf Shell
```

[open](#) build/pdf/myproject.pdf

15.9 Next Steps

- Explore Advanced Examples for more complex examples
- Read Configuration for all options
- Learn about Templates for customization

16 Advanced Examples

Advanced configurations and use cases for typsphinx.

16.1 Using Typst Universe Packages

16.1.1 charged-ieee Template

Use the charged-ieee package for IEEE-style papers:

conf.py:

```
project = "Machine Learning Applications"
author = "John Doe"

# Use IEEE package
typst_package = "@preview/charged-ieee:0.1.4"

# Configure template with parameters
ieee_abstract = """
This paper presents novel approaches to machine learning
applications in computer vision.
"""

ieee_keywords = ["Machine Learning", "Computer Vision", "AI"]

typst_template_function = {
    "name": "ieee",
    "params": {
        "abstract": ieee_abstract,
        "index-terms": ieee_keywords,
        "paper-size": "us-letter",
    }
}

# Detailed author information
typst_authors = {
    "John Doe": {
        "department": "Computer Science",
        "organization": "MIT",
        "email": "john@mit.edu"
    }
}
```

16.1.2 modern-cv Template

Create a CV/resume with modern-cv:

```
typst_package = "@preview/modern-cv:0.7.0"
```

 Python


```
typst_template_function = {  
  "name": "modern-cv",  
  "params": {  
    "name": "John Doe",  
    "job-title": "Software Engineer",  
    "email": "john@example.com",  
    "github": "johndoe",  
  }  
}
```

16.2 Custom Template Wrapping

Wrap external packages with custom logic:

_templates/custom_ieee.typ:

```
#import "@preview/charged-ieee:0.1.4": ieee
```

 Typst


```
#let project(  
  title: "",  
  authors: (),  
  date: none,  
  body  
) = {  
  // Transform simple author tuples to IEEE format  
  let ieee_authors = authors.map(name => (  
    name: name,  
    department: "Engineering",  
    organization: "My Organization",  
    location: "City, State",  
    email: name.split(" ").at(0).lower() + "@example.com"  
  ))  
  
  // Define abstract and keywords (could be parameters)  
  let ieee_abstract = [  
    This document demonstrates custom template wrapping  
    with automatic author transformation.  
  ]  
  
  let ieee_keywords = (  
    "Documentation",  
    "Typst",  
    "Automation"  
  )  
}
```

```
// Apply IEEE template
show: ieee.with(
  title: title,
  authors: ieee_authors,
  abstract: ieee_abstract,
  index-terms: ieee_keywords,
  bibliography: "refs.bib",
)

body
}
```

conf.py:

```
typst_template = "_templates/custom_ieee.typ"
typst_package = "@preview/charged-ieee:0.1.4"
```


 Python

16.3 Multi-Document Projects

Build multiple related documents:

conf.py:

```
typst_documents = [
  # (source, output, title, author, class)
  ("index", "main", "Main Documentation", "Team", "typst"),
  ("api/index", "api-reference", "API Reference", "Team", "typst"),
  ("tutorial/index", "tutorial", "Tutorial", "Team", "typst"),
]
```

 Python

Each document is built separately with its own output file.


16.4 Custom Styling

Apply custom fonts and colors:

_templates/styled.typ:

```
#let project(
  title: "",
  primary-color: blue,
  body
) = {
  // Set custom font
  set text(
    font: "New Computer Modern",
    size: 11pt,
  )

  // Custom heading style
```

 Typst


```

show heading.where(level: 1): it => {
  set text(fill: primary-color, size: 20pt, weight: "bold")
  it
  v(0.5em)
}

show heading.where(level: 2): it => {
  set text(fill: primary-color.lighten(20%), size: 16pt)
  it
  v(0.3em)
}

// Title page
align(center)[
  #text(size: 28pt, fill: primary-color, weight: "bold")[
    #title
  ]
]

pagebreak()

body
}

```

conf.py:

```

typst_template = "_templates/styled.typ"
typst_template_function = {
  "name": "project",
  "params": {
    "primary-color": "rgb(0, 102, 204)", # Custom blue
  }
}

```

16.5 Conditional Content

Use different templates for different documents:

conf.py:

```

# Default template for most documents
typst_template = "_templates/default.typ"

# Define multiple documents
typst_documents = [
  ("index", "main", "Main Docs", "Team", "typst"),
  ("paper", "research-paper", "Research Paper", "Authors", "typst"),
]

```

For document-specific templates, you can use Sphinx's per-file configuration or conditional logic in your template.

16.6 Bibliographies

Include bibliographies with BibTeX:

index.rst:

Research Paper	rst
=====	
According to Smith et al. [Smith2023]_, machine learning...	
References	

.. [Smith2023] Smith, J. (2023). Machine Learning Advances.	
Journal of AI Research, 15(2), 123-145.	

Custom template with bibliography:

<pre>#let project(title: "", body) = { set page(paper: "us-letter") text(20pt, weight: "bold")[#title] v(2em) body // Bibliography section pagebreak() heading(numbering: none)[References] // Bibliography rendered from .bib file }</pre>	Typst
--	-------

16.7 Advanced Math

Complex mathematical expressions:

index.rst:

Advanced Mathematics	rst
=====	
Matrix equation:	
.. math::	
$\mathbf{A} \mathbf{x} = \mathbf{b}$	

```

\\begin{pmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\\end{pmatrix}
\\begin{pmatrix}
x_1 \\
x_2
\\end{pmatrix}
=
\\begin{pmatrix}
b_1 \\
b_2
\\end{pmatrix}

```

Aligned equations:

```

.. math::

\\begin{align}
f(x) &= x^2 + 2x + 1 \\
&= (x + 1)^2
\\end{align}

```

16.8 CI/CD Integration

GitHub Actions workflow for documentation:

.github/workflows/docs.yml:

```

name: Documentation

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  build-docs:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Python
        uses: actions/setup-python@v5
        with:
          python-version: "3.11"

      - name: Install dependencies
        run: |

```

YAML

```

    pip install -e .
    pip install sphinx furo sphinx-autodoc-typehints

- name: Build HTML documentation
  run: |
    cd docs
    sphinx-build -b html source _build/html

- name: Build PDF documentation
  run: |
    cd docs
    sphinx-build -b typstpdf source _build/pdf

- name: Upload PDF artifact
  uses: actions/upload-artifact@v4
  with:
    name: documentation-pdf
    path: docs/_build/pdf/*.pdf

```

16.9 Performance Optimization

For large documentation projects:

conf.py:


```

# Parallel build
import multiprocessing
parallel_read_safe = True
parallel_write_safe = True

# Limit depth for faster builds
typst_documents = [
    ("index", "output", "Title", "Author", "typst"),
]

# Disable expensive features if not needed
typst_use_codly = True # Keep code highlighting
typst_code_line_numbers = False # Disable if not needed

```

 Python

16.10 See Also

- Configuration - All configuration options
- Templates - Template system details
- Typst Documentation - Official Typst reference
- Typst Universe - Package repository

17 API Reference

This section provides detailed API documentation for typsphinx.

17.1 Builders

Typst builder for Sphinx.

This module implements the `TypstBuilder` class, which is responsible for building Typst output from Sphinx documentation.

`class sphinx.builder.TypstBuilder(app, env)`

Bases: `Builder`

Builder class for Typst output format.

This builder converts Sphinx documentation to Typst markup files (`.typ`), which can then be compiled to PDF using the Typst compiler.

Parameters:

- **`app`** (*Sphinx*)
- **`env`** (*BuildEnvironment*)

`name:str='typst'`

The builder's name.

This is the value used to select builders on the command line.

`format:str='typst'`

The builder's output format, or '' if no document output is produced.

This is commonly the file extension, e.g. "html", though any string value is accepted.

The builder's format string can be used by various components such as `SphinxPostTransform` or extensions to determine their compatibility with the builder.

`out_suffix='.typ'allow_parallel:bool=True`

Whether it is safe to make parallel `write_doc()` calls.

`init()`

Initialize the builder.

This method is called once at the beginning of the build process.

Return type:

None

`get_outdated_docs()`

Return an iterator of document names that need to be rebuilt.

For now, we rebuild all documents on every build.

Return type:

Iterator[str]

Returns:

Iterator of document names that are outdated

`get_target_uri(docname, typ=None)`

Return the target URI for a document.

Parameters:

- **docname** (str) – Name of the document
- **typ** (Optional[str]) – Type of the target (not used for Typst builder)

Return type:

str

Returns:

Target URI string

prepare_writing(docnames)

Prepare for writing the documents.

This method is called before writing begins.

Writes the template file to the output directory for master documents to import.

Parameters:

docnames (Set[str]) – Set of document names to be written

Return type:

None

write(build_docnames, updated_docnames, method='update')

Override write() to preserve toctree nodes.

By default, Sphinx's Builder.write() calls env.get_and_resolve_doctree() which expands toctree nodes into compact_paragraph with links.

For Typst, we need the original toctree nodes to generate #include() directives.

This method uses env.get_doctree() instead to preserve toctree nodes.

Parameters:

- **build_docnames** (Optional[Set[str]]) – Document names to build (None = all)
- **updated_docnames** (Set[str]) – Document names that were updated
- **method** (str) – Build method ('update' or 'all')

Return type:

None

post_process_images(doctree)

Post-process images in the document tree.

Collects all image nodes from the document tree and tracks them in self.images dictionary for later copying to the output directory.

Parameters:

doctree (document) – Document tree to process

Return type:

None

write_doc(docname, doctree)

Write a document.

This method is called for each document that needs to be written.

Requirement 13.1: `reStructuredText` `.typ`

Requirement 13.12:

Parameters:

- **docname** (str) – Name of the document
- **doctree** (document) – Document tree to be written

Return type:

None

copy_image_files()

Copy image files to the output directory.

Iterates through all tracked images and copies them from the source directory to the output directory, preserving relative paths.

Return type:

None

finish()

Finish the build process.

This method is called once after all documents have been written.
Copies image files to the output directory.

Return type:

None

class sphinx.builder.TypstPDFBuilder(app, env)

Bases: `TypstBuilder`

Builder class for generating PDF output directly from Typst.

This builder extends `TypstBuilder` to compile generated `.typ` files to PDF using the `typst-py` package.

Requirement 9.3: `TypstPDFBuilder` extends `TypstBuilder`

Requirement 9.4: Generate PDF from Typst markup

Parameters:

- **app** (*Sphinx*)
- **env** (*BuildEnvironment*)

name: str = 'typstpdf'

The builder's name.

This is the value used to select builders on the command line.

format: str = 'pdf'

The builder's output format, or '' if no document output is produced.
This is commonly the file extension, e.g. "html",
though any string value is accepted.

The builder's format string can be used by various components such as SphinxPostTransform or extensions to determine their compatibility with the builder.

out_suffix='.pdf'**write_doc(docname, doctree)**

Write a document as both .typ and .pdf.

Override to generate .typ file (not .pdf) during the write phase. The .pdf will be generated in finish() by compiling the .typ file.

Parameters:

- **docname** (str) – Name of the document
- **doctree** (document) – Document tree to be written

Return type:

None

finish()

Finish the build process by compiling Typst files to PDF.

After the parent TypstBuilder has generated .typ files, this method compiles them to PDF using typst-py.

Only master documents (defined in typst_documents) are compiled to PDF. Included documents are not compiled individually.

Requirement 9.2: Execute Typst compilation within Python

Requirement 9.4: Generate PDF from Typst markup

Return type:

None

PDF generation utilities using typst-py.

This module provides functionality for generating PDFs from Typst markup using the typst Python package (Requirement 9).

exceptiontypsphinx.pdf.TypstCompilationError(message, typst_error=None, source_location=None)

Bases: Exception

Exception raised when Typst compilation fails.

This exception provides detailed information about compilation errors, including the original error from typst-py and contextual information.

Parameters:

- **message** (str)
- **typst_error** (Exception | None)
- **source_location** (str | None)

message

Human-readable error message

typst_error

Original error from typst compiler

source_location

Location information if available

Requirement 10.3: Error detection and handling

Requirement 10.4: Error message parsing and user display

typsphinx.pdf.check_typst_available()

Check if typst package is available.

Raises:

ImportError – If typst package is not installed

Return type:

None

Requirement 9.1: Typst compiler functionality as dependency

Requirement 9.7: Automatic availability of Typst compiler

Return type:

None

typsphinx.pdf.get_typst_version()

Get the version of the typst package.

Return type:

str

Returns:

Version string (e.g., “0.13.7”)

Requirement 9.7: Version information for Typst compiler

typsphinx.pdf.compile_typst_to_pdf(typst_content, root_dir=None)

Compile Typst content to PDF bytes.

Parameters:

- **typst_content** (str) – Typst markup content
- **root_dir** (Optional[str]) – Root directory for resolving includes and images

Return type:

bytes

Returns:

PDF content as bytes

Raises:

- **ImportError** – If typst package not available
- **TypstCompilationError** – If compilation fails

Requirement 9.2: Execute Typst compilation within Python environment

Requirement 9.4: Generate PDF from Typst markup

Requirement 10.3: Error detection and handling

17.2 Writer and Translator

Typst writer for docutils.

This module implements the `TypstWriter` class, which converts docutils document trees to Typst markup.

`class typstphinx.writer.TypstWriter(builder)`

Bases: `Writer`

Writer class for Typst output format.

This writer converts docutils document trees to Typst markup files.

Parameters:

`builder` (*Any*)

`supported`=('typst',)

Formats this writer supports.

`translate()`

Translate the document tree to Typst markup.

This method creates a `TypstTranslator` and visits the document tree, then wraps the output with a template using `TemplateEngine`.

For master documents (defined in `typst_documents`), the full template is applied. For included documents, only the body content is output.

Return type:

None

Typst translator for docutils nodes.

This module implements the `TypstTranslator` class, which translates docutils nodes to Typst markup.

`class typstphinx.translator.TypstTranslator(document, builder)`

Bases: `SphinxTranslator`

Translator class that converts docutils nodes to Typst markup.

This translator visits nodes in the document tree and generates corresponding Typst markup.

Parameters:

- **`document`** (*document*)
- **`builder`** (*Any*)

`astext()`

Return the translated text as a string.

Return type:

`str`

Returns:

The translated Typst markup

add_text(text)

Add text to the output body or table cell content.

Parameters:

text (str) – The text to add

Return type:

None

visit_document(node)

Visit a document node.

Generates opening code block wrapper for unified code mode.

Parameters:

node (document) – The document node

Return type:

None

depart_document(node)

Depart a document node.

Generates closing code block wrapper for unified code mode.

Parameters:

node (document) – The document node

Return type:

None

visit_section(node)

Visit a section node.

Parameters:

node (section) – The section node

Return type:

None

depart_section(node)

Depart a section node.

Parameters:

node (section) – The section node

Return type:

None

visit_title(node)

Visit a title node.

Generates heading() function call with level parameter.
Child text nodes will be wrapped in text() automatically.

Parameters:

node (title) – The title node

Return type:

None

depart_title(node)

Depart a title node.

Closes heading() function call.

Parameters:

node (title) – The title node

Return type:

None

visit_subtitle(node)

Visit a subtitle node.

Generates emph() function for subtitle (no # prefix in code mode).
Child text nodes will be wrapped in text() automatically.

Parameters:

node (subtitle) – The subtitle node

Return type:

None

depart_subtitle(node)

Depart a subtitle node.

Closes emph() function.

Parameters:

node (subtitle) – The subtitle node

Return type:

None

visit_compound(node)

Visit a compound node.

Compound nodes are containers that group related content.
They are often used to wrap toctree directives.

Parameters:

node (compound) – The compound node

Return type:

None

depart_compound(node)

Depart a compound node.

Parameters:

node (compound) – The compound node

Return type:

None

visit_container(node)

Visit a container node.

Handle Sphinx-generated containers, particularly literal-block-wrapper for captioned code blocks (Issue #20).

Parameters:

node (container) – The container node

Return type:

None

depart_container(node)

Depart a container node.

Parameters:

node (container) – The container node

Return type:

None

visit_paragraph(node)

Visit a paragraph node.

Wraps paragraph content in par() function for unified code mode.

Code mode doesn't auto-recognize paragraph breaks from blank lines.

Exception: Inside list items, paragraphs are not wrapped in par() to avoid syntax like “- par(text(...))” which is invalid.

Parameters:

node (paragraph) – The paragraph node

Return type:

None

depart_paragraph(node)

Depart a paragraph node.

Closes par({}) function and adds spacing.

Parameters:

node (paragraph) – The paragraph node

Return type:

None

visit_comment(node)

Visit a comment node.

Comments are skipped entirely in Typst output as they are meant for source-level documentation only.

Parameters:

node (comment) – The comment node

Raises:

nodes.SkipNode – Always raised to skip the comment

Return type:

None

depart_comment(node)

Depart a comment node.

Parameters:

node (comment) – The comment node

Return type:

None

i Info

```
par({text("This method is not called when SkipNode is raised in visit_comment.")})
```

visit_raw(node)

Visit a raw node.

Pass through content if format is 'typst', otherwise skip.

Parameters:

node (raw) – The raw node

Raises:

nodes.SkipNode – When format is not 'typst'

Return type:

None

depart_raw(node)

Depart a raw node.

Parameters:

node (raw) – The raw node

Return type:

None

i Info

```
par({text("This method is not called when SkipNode is raised in visit_raw.")})
```

visit_Text(node)

Visit a text node.

Wraps text in text() function for unified code mode.

Uses string escaping (not markup escaping).

Exception: Inside literal blocks, text is output directly without text() wrapping to preserve code content.

Parameters:

node (Text) – The text node

Return type:

None

depart_Text(node)

Depart a text node.

Parameters:

node (Text) – The text node

Return type:

None

visit_emphasis(node)

Visit an emphasis (italic) node.

Generates emph() function call. Child text nodes will be wrapped in text() automatically.

Parameters:

node (emphasis) – The emphasis node

Return type:

None

depart_emphasis(node)

Depart an emphasis (italic) node.

Closes emph({}) function call.

Parameters:

node (emphasis) – The emphasis node

Return type:

None

visit_strong(node)

Visit a strong (bold) node.

Generates strong() function call. Child text nodes will be wrapped in text() automatically.

Parameters:

node (strong) – The strong node

Return type:

None

depart_strong(node)

Depart a strong (bold) node.

Closes strong({}) function call.

Parameters:

node (strong) – The strong node

Return type:

None

visit_literal(node)

Visit a literal (inline code) node.

Generates raw() function call with backtick raw string.

Uses backticks to avoid escaping issues.

Parameters:

node (literal) – The literal node

Return type:

None

depart_literal(node)

Depart a literal (inline code) node.

This is not called when SkipNode is raised in visit_literal.

Parameters:

node (literal) – The literal node

Return type:

None

visit_subscript(node)

Visit a subscript node.

Generates sub() function call. Child text nodes will be wrapped in text() automatically.

Parameters:

node (subscript) – The subscript node

Return type:

None

depart_subscript(node)

Depart a subscript node.

Closes sub() function call.

Parameters:

node (subscript) – The subscript node

Return type:

None

visit_superscript(node)

Visit a superscript node.

Generates super() function call. Child text nodes will be wrapped in text() automatically.

Parameters:

node (superscript) – The superscript node

Return type:

None

depart_superscript(node)

Depart a superscript node.

Closes super() function call.

Parameters:

node (superscript) – The superscript node

Return type:

None

visit_bullet_list(node)

Visit a bullet list node.

Outputs list(and prepares for stream-based item rendering.

Parameters:

node (bullet_list) – The bullet list node

Return type:

None

depart_bullet_list(node)

Depart a bullet list node.

Closes the list() function.

Parameters:

node (bullet_list) – The bullet list node

Return type:

None

visit_enumerated_list(node)

Visit an enumerated (numbered) list node.

Outputs enum() and prepares for stream-based item rendering.

Parameters:

node (enumerated_list) – The enumerated list node

Return type:

None

depart_enumerated_list(node)

Depart an enumerated (numbered) list node.

Closes the enum() function.

Parameters:

node (enumerated_list) – The enumerated list node

Return type:

None

visit_list_item(node)

Visit a list item node.

Adds comma separator if not first item, then prepares for item content.

Parameters:

node (list_item) – The list item node

Return type:

None

depart_list_item(node)

Depart a list item node.

Close the {} block wrapper and mark that we're no longer in a list item.

Parameters:

node (list_item) – The list item node

Return type:

None

visit_literal_block(node)

Visit a literal block (code block) node.

Implements Task 4.2.2: codly forced usage with #codly-range() for highlighted lines

Design 3.5: All code blocks use codly, with #codly-range() for highlights

Requirements 7.3, 7.4: Support line numbers and highlighted lines

Issue #20: Support :linenos:, :caption:, and :name: options

Issue #31: Support :lineno-start: and :dedent: options

Parameters:

node (literal_block) – The literal block node

Return type:

None

depart_literal_block(node)

Depart a literal block (code block) node.

Issue #20: Handle closing figure bracket and labels.

Parameters:

node (literal_block) – The literal block node

Return type:

None

visit_definition_list(node)

Visit a definition list node.

Collects all term-definition pairs and generates terms() function in unified code mode.

Parameters:

node (definition_list) – The definition list node

Return type:

None

depart_definition_list(node)

Depart a definition list node.

Generates terms() function with all collected term-definition pairs.

Parameters:

node (definition_list) – The definition list node

Return type:

None

visit_definition_list_item(node)

Visit a definition list item node.

Parameters:

node (definition_list_item) – The definition list item node

Return type:

None

depart_definition_list_item(node)

Depart a definition list item node.

Parameters:

node (definition_list_item) – The definition list item node

Return type:

None

visit_term(node)

Visit a term (definition list term) node.

Starts buffering term content.

Parameters:

node (term) – The term node

Return type:

None

depart_term(node)

Depart a term (definition list term) node.

Saves buffered term content.

Parameters:

node (term) – The term node

Return type:

None

visit_definition(node)

Visit a definition (definition list definition) node.

Starts buffering definition content.

Parameters:

node (definition) – The definition node

Return type:

None

depart_definition(node)

Depart a definition (definition list definition) node.

Saves buffered definition content and pairs it with the term.

Parameters:

node (definition) – The definition node

Return type:

None

visit_figure(node)

Visit a figure node.

Generates figure() function call (no # prefix in code mode).

Parameters:

node (figure) – The figure node

Return type:

None

depart_figure(node)

Depart a figure node.

Parameters:

node (figure) – The figure node

Return type:

None

visit_caption(node)

Visit a caption node.

Handles captions for both figures and code blocks (Issue #20).

Parameters:

node (caption) – The caption node

Return type:

None

depart_caption(node)

Depart a caption node.

Parameters:

node (caption) – The caption node

Return type:

None

visit_table(node)

Visit a table node.

Parameters:

node (table) – The table node

Return type:

None

depart_table(node)

Depart a table node.

Parameters:

node (table) – The table node

Return type:

None

visit_tgroup(node)

Visit a tgroup (table group) node.

Parameters:

node (tgroup) – The tgroup node

Return type:

None

depart_tgroup(node)

Depart a tgroup (table group) node.

Parameters:

node (tgroup) – The tgroup node

Return type:

None

visit_colspec(node)

Visit a colspec (column specification) node.

Parameters:

node (colspec) – The colspec node

Return type:

None

depart_colspec(node)

Depart a colspec (column specification) node.

Parameters:

node (colspec) – The colspec node

Return type:

None

visit_thead(node)

Visit a thead (table header) node.

Parameters:

node (thead) – The thead node

Return type:

None

depart_thead(node)

Depart a thead (table header) node.

Parameters:

node (thead) – The thead node

Return type:

None

visit_tbody(node)

Visit a tbody (table body) node.

Parameters:

node (tbody) – The tbody node

Return type:

None

depart_tbody(node)

Depart a tbody (table body) node.

Parameters:

node (tbody) – The tbody node

Return type:

None

visit_row(node)

Visit a row (table row) node.

Parameters:

node (row) – The row node

Return type:

None

depart_row(node)

Depart a row (table row) node.

Parameters:

node (row) – The row node

Return type:

None

visit_entry(node)

Visit an entry (table cell) node.

Parameters:

node (entry) – The entry node

Return type:

None

depart_entry(node)

Depart an entry (table cell) node.

Parameters:

node (entry) – The entry node

Return type:

None

visit_block_quote(node)

Visit a block quote node.

Generates quote() function call (no # prefix in code mode).

Parameters:

node (block_quote) – The block quote node

Return type:

None

depart_block_quote(node)

Depart a block quote node.

Parameters:

node (block_quote) – The block quote node

Return type:

None

visit_attribution(node)

Visit an attribution node (quote attribution).

Parameters:

node (attribution) – The attribution node

Return type:

None

depart_attribution(node)

Depart an attribution node.

Parameters:

node (attribution) – The attribution node

Return type:

None

visit_image(node)

Visit an image node.

Generates image() function call (no # prefix in code mode).

Parameters:

node (image) – The image node

Return type:

None

depart_image(node)

Depart an image node.

Parameters:

node (image) – The image node

Return type:

None

visit_target(node)

Visit a target node (label definition).

Parameters:

node (target) – The target node

Return type:

None

depart_target(node)

Depart a target node.

Parameters:

node (target) – The target node

Return type:

None

visit_pending_xref(node)

Visit a pending_xref node (Sphinx cross-reference).

Parameters:

node (Node) – The pending_xref node

Return type:

None

depart_pending_xref(node)

Depart a pending_xref node.

Parameters:

node (Node) – The pending_xref node

Return type:

None

visit_toctree(node)

Visit a toctree node (Sphinx table of contents tree).

Requirement 13: Multi-document integration and toctree processing

- Generate #include() for each entry
- Apply #set heading(offset: 1) to lower heading levels
- Issue #5: Fix relative paths for nested toctrees

```
“list({ text(“Calculate relative paths from current document”) })
”
```

- Issue #7: Simplify toctree output with single content block
 - Generate single #[...] block containing all includes
 - Apply #set heading(offset: 1) once per toctree

Parameters:

node (Node) – The toctree node

Return type:

None

Notes

This method generates Typst `#include()` directives for each toctree entry within a single content block `#[...]` to apply heading offset without displaying the block delimiters in the output. This simplifies the generated Typst code and improves readability.

depart_toctree(node)

Depart a toctree node.

Parameters:

node (Node) – The toctree node

Return type:

None

visit_reference(node)

Visit a reference node (link).

Generates `link()` function call (no `#` prefix in code mode).

Parameters:

node (reference) – The reference node

Return type:

None

depart_reference(node)

Depart a reference node.

Parameters:

node (reference) – The reference node

Return type:

None

unknown_visit(node)

Handle unknown nodes during visit.

Parameters:

node (Node) – The unknown node

Return type:

None

unknown_departure(node)

Handle unknown nodes during departure.

Parameters:

node (Node) – The unknown node

Return type:

None

visit_math(node)

Visit an inline math node.

Implements Task 6.2: LaTeX math conversion (mitex)

Implements Task 6.3: Labeled equations

Implements Task 6.4: Typst native math support

Implements Task 6.5: Math fallback functionality

Requirement 4.3: Inline math should use #mi(...) format (LaTeX)

Requirement 4.9: Fallback when typst_use_mitex=False

Requirement 5.2: Inline math should use \$...\$ format (Typst native)

Requirement 4.7: Labeled equations should generate <eq:label> format

Design 3.3: Support both mitex and Typst native math

Parameters:

node (math) – The inline math node

Return type:

None

depart_math(node)

Depart an inline math node.

Parameters:

node (math) – The inline math node

Return type:

None

visit_math_block(node)

Visit a block math node.

Implements Task 6.2: LaTeX math conversion (mitex)

Implements Task 6.3: Labeled equations

Implements Task 6.4: Typst native math support

Implements Task 6.5: Math fallback functionality

Requirement 4.2: Block math should use #mitex(...) format (LaTeX)

Requirement 4.9: Fallback when typst_use_mitex=False

Requirement 5.2: Block math should use \$... \$ format (Typst native)

Requirement 4.7: Labeled equations should generate <eq:label> format

Design 3.3: Support both mitex and Typst native math

Parameters:

node (*math_block*) – The block math node

Return type:

None

depart_math_block(node)

Depart a block math node.

Parameters:

node (*math_block*) – The block math node

Return type:

None

visit_note(node)

Visit a note admonition (converts to #info[]).

Return type:

None

Parameters:

node (*note*)

depart_note(node)

Depart a note admonition.

Return type:

None

Parameters:

node (*note*)

visit_warning(node)

Visit a warning admonition (converts to #warning[]).

Return type:

None

Parameters:

node (*warning*)

depart_warning(node)

Depart a warning admonition.

Return type:

None

Parameters:

node (*warning*)

visit_tip(node)

Visit a tip admonition (converts to #tip[]).

Return type:

None

Parameters:

node (*tip*)

depart_tip(node)

Depart a tip admonition.

Return type:

None

Parameters:

node (*tip*)

visit_important(node)

Visit an important admonition (converts to #warning(title: "Important")[]).

Return type:

None

Parameters:

node (*important*)

depart_important(node)

Depart an important admonition.

Return type:

None

Parameters:

node (*important*)

visit_caution(node)

Visit a caution admonition (converts to #warning[]).

Return type:

None

Parameters:

node (*caution*)

depart_caution(node)

Depart a caution admonition.

Return type:

None

Parameters:

node (*caution*)

visit_seealso(node)

Visit a seealso admonition (converts to #info(title: “See Also”)[]).

Return type:

None

Parameters:

node (*seealso*)

depart_seealso(node)

Depart a seealso admonition.

Return type:

None

Parameters:

node (*seealso*)

visit_inline(node)

Visit an inline node.

Inline nodes are generic containers for inline content.

They are often used for cross-references with specific CSS classes.

Task 7.4: Handle inline nodes, especially those with ‘xref’ class

Requirement 3.1: Cross-references and links

Return type:

None

Parameters:

node (*inline*)

depart_inline(node)

Depart an inline node.

Return type:

None

Parameters:

node (*inline*)

visit_index(node)

Visit an index node.

Index entries are skipped in Typst/PDF output as we don’t generate indices.

Return type:

None

Parameters:

node (*index*)

depart_index(node)

Depart an index node.

Return type:

None

Parameters:

node (*index*)

visit_desc(node)

Visit a desc node (API description container).

Desc nodes contain API descriptions (classes, functions, methods, etc.).

Return type:

None

Parameters:

node (*desc*)

depart_desc(node)

Depart a desc node.

Add spacing after API description blocks.

Return type:

None

Parameters:

node (*desc*)

visit_desc_signature(node)

Visit a desc_signature node (API element signature).

Signatures are rendered in bold using `strong({})` wrapper.

Return type:

None

Parameters:

node (*desc_signature*)

depart_desc_signature(node)

Depart a desc_signature node.

Return type:

None

Parameters:

node (*desc_signature*)

visit_desc_content(node)

Visit a desc_content node (API description content).

Return type:

None

Parameters:

node (*desc_content*)

depart_desc_content(node)

Depart a desc_content node.

Return type:

None

Parameters:

node (*desc_content*)

visit_desc_annotation(node)

Visit a desc_annotation node (type annotations like 'class', 'async', etc.).

Return type:

None

Parameters:

node (*desc_annotation*)

depart_desc_annotation(node)

Depart a desc_annotation node.

Space after annotation is handled by desc_sig_space node.

Return type:

None

Parameters:

node (*desc_annotation*)

visit_desc_addname(node)

Visit a desc_addname node (module name prefix).

Return type:

None

Parameters:

node (*desc_addname*)

depart_desc_addname(node)

Depart a desc_addname node.

Return type:

None

Parameters:

node (*desc_addname*)

visit_desc_name(node)

Visit a desc_name node (function/class name).

Return type:

None

Parameters:

node (*desc_name*)

depart_desc_name(node)

Depart a desc_name node.

Return type:

None

Parameters:

node (*desc_name*)

visit_desc_parameterlist(node)

Visit a desc_parameterlist node (parameter list container).

Parameters are concatenated with + inside text parentheses.

Return type:

None

Parameters:

node (*desc_parameterlist*)

depart_desc_parameterlist(node)

Depart a desc_parameterlist node.

Return type:

None

Parameters:

node (*desc_parameterlist*)

visit_desc_parameter(node)

Visit a desc_parameter node (individual parameter).

Return type:

None

Parameters:

node (*desc_parameter*)

depart_desc_parameter(node)

Depart a desc_parameter node.

Add comma + space between parameters if not last.

Return type:

None

Parameters:

node (*desc_parameter*)

visit_field_list(node)

Visit a field_list node (structured fields like Parameters, Returns).

Return type:

None

Parameters:

node (*field_list*)

depart_field_list(node)

Depart a field_list node.

Add spacing after field lists.

Return type:

None

Parameters:

node (*field_list*)

visit_field(node)

Visit a field node (individual field in a field list).

Return type:

None

Parameters:

node (*field*)

depart_field(node)

Depart a field node.

Return type:

None

Parameters:

node (*field*)

visit_field_name(node)

Visit a field_name node (field name like 'Parameters', 'Returns').

Field names are rendered in bold with a colon (no # prefix in code mode).

Return type:

None

Parameters:

node (*field_name*)

depart_field_name(node)

Depart a field_name node.

Return type:

None

Parameters:

node (*field_name*)

visit_field_body(node)

Visit a field_body node (field content).

Return type:

None

Parameters:

node (*field_body*)

depart_field_body(node)

Depart a field_body node.

Add newline after field body.

Return type:

None

Parameters:

node (*field_body*)

visit_rubric(node)

Visit a rubric node (section subheading).

Rubrics are rendered as subsection headings using strong({}) wrapper.

Return type:

None

Parameters:

node (*rubric*)

depart_rubric(node)

Depart a rubric node.

Return type:

None

Parameters:

node (*rubric*)

visit_title_reference(node)

Visit a `title_reference` node (reference to a title).

Title references are rendered in emphasis using `emph({})` wrapper.

Return type:

None

Parameters:

node (*title_reference*)

depart_title_reference(node)

Depart a `title_reference` node.

Return type:

None

Parameters:

node (*title_reference*)

visit_desc_sig_keyword(node)

Visit a `desc_sig_keyword` node (keywords in signatures like ‘class’, ‘def’).

Return type:

None

Parameters:

node (*desc_sig_keyword*)

depart_desc_sig_keyword(node)

Depart a `desc_sig_keyword` node.

Return type:

None

Parameters:

node (*desc_sig_keyword*)

visit_desc_sig_space(node)

Visit a `desc_sig_space` node (whitespace in signatures).

Return type:

None

Parameters:

node (*desc_sig_space*)

depart_desc_sig_space(node)

Depart a `desc_sig_space` node.

Return type:

None

Parameters:

node (*desc_sig_space*)

visit_desc_sig_name(**node**)

Visit a desc_sig_name node (names in signatures).

Return type:

None

Parameters:

node (*desc_sig_name*)

depart_desc_sig_name(**node**)

Depart a desc_sig_name node.

Return type:

None

Parameters:

node (*desc_sig_name*)

visit_desc_sig_punctuation(**node**)

Visit a desc_sig_punctuation node (punctuation in signatures like ':', '=').

Return type:

None

Parameters:

node (*desc_sig_punctuation*)

depart_desc_sig_punctuation(**node**)

Depart a desc_sig_punctuation node.

Return type:

None

Parameters:

node (*desc_sig_punctuation*)

visit_desc_sig_operator(**node**)

Visit a desc_sig_operator node (operators in signatures).

Return type:

None

Parameters:

node (*desc_sig_operator*)

depart_desc_sig_operator(**node**)

Depart a desc_sig_operator node.

Return type:

None

Parameters:

node (*desc_sig_operator*)

visit_literal_strong(node)

Visit a literal_strong node (bold literal text in field lists).

Return type:

None

Parameters:

node (*inline*)

depart_literal_strong(node)

Depart a literal_strong node.

Return type:

None

Parameters:

node (*inline*)

visit_literal_emphasis(node)

Visit a literal_emphasis node (emphasized literal text in field lists).

Return type:

None

Parameters:

node (*inline*)

depart_literal_emphasis(node)

Depart a literal_emphasis node.

Return type:

None

Parameters:

node (*inline*)

17.3 Template Engine

Template engine for Typst document generation.

This module implements template loading, parameter mapping, and rendering for Typst documents (Requirement 8).

classtypstphnix.template_engine.TemplateEngine(template_path=None, template_name=None, search_paths=None, parameter_mapping=None, typst_package=None, typst_template_function=None, typst_package_imports=None, typst_authors=None, typst_author_params=None)

Bases: object

Manages Typst templates for document generation.

Responsibilities:

- Load default or custom Typst templates
- Search templates in multiple directories with priority
- Provide fallback to default template when custom template not found
- Map Sphinx metadata to template parameters
- Render final Typst document with template and content

Requirement 8.1: Default Typst template included in package

Requirement 8.2: Support custom template specification

Requirement 8.7: Priority search in user project directory

Requirement 8.9: Fallback to default template with warning

Parameters:

- **template_path** (*str* | *None*)
- **template_name** (*str* | *None*)
- **search_paths** (*List[str]* | *None*)
- **parameter_mapping** (*Dict[str, str]* | *None*)
- **typst_package** (*str* | *None*)
- **typst_template_function** (*Any* | *None*)
- **typst_package_imports** (*List[str]* | *None*)
- **typst_authors** (*Dict[str, Dict[str, Any]]* | *None*)
- **typst_author_params** (*Dict[str, Dict[str, Any]]* | *None*)

DEFAULT_PARAMETER_MAPPING={'author': 'authors', 'project': 'title', 'release': 'date'}
get_default_template_path()

Get the path to the default template bundled with the package.

Return type:

str

Returns:

Absolute path to default template file

load_template()

Load Typst template with priority order:

1. Explicit template_path if provided
2. Search for template_name in search_paths (first match wins)
3. Default template bundled with package

Return type:

str

Returns:

Template content as string

Requirement 8.1: Load default template

Requirement 8.2: Load custom template

Requirement 8.7: Search in user project directory

Requirement 8.9: Fallback to default with warning

map_parameters(sphinx_metadata)

Map Sphinx metadata to template parameters.

Parameters:

sphinx_metadata (Dict[str, Any]) – Dictionary of Sphinx configuration metadata (project, author, release, etc.)

Return type:

Dict[str, Any]

Returns:

Dictionary of template parameters ready to pass to template

Requirement 8.3: Pass Sphinx metadata to template

Requirement 8.4: Support different parameter names

Requirement 8.5: Standard metadata name transformation

Requirement 8.8: Convert to arrays and complex structures

generate_package_import()

Generate Typst package import statement.

Return type:

str

Returns:

Import statement string, or empty string if no package specified

Requirement 8.6: Typst Universe external template packages

extract_toc_tree_options(doctree)

Extract toctree options from doctree for template parameters.

Parameters:

doctree (Any) – Docutils document tree

Return type:

Dict[str, Any]

Returns:

Dictionary of toctree options for template

Requirement 8.12: toctree options passed as template parameters

Requirement 8.13: template reflects toctree options in #outline()

Requirement 13.8: #outline() managed at template level

Requirement 13.9: toctree options mapped to template parameters

get_template_content()

Get the template content for writing to a separate file.

Return type:

str

Returns:

Template content as string

This is used when templates are written as separate files instead of being inlined in the main document.

render(params, body, template_file=None)

Render final Typst document with template and body.

Parameters:

- **params** (Dict[str, Any]) – Template parameters (title, authors, etc.)
- **body** (str) – Document body content (Typst markup)
- **template_file** (str) – Path to template file for import (relative to output dir).
If None, template is inlined (old behavior).
If specified, template is imported from file.

Return type:

str

Returns:

Complete Typst document string

Requirement 8.2: Use custom template

Requirement 8.10: Pass document settings to template

Requirement 8.14: #outline() in template, not body

17.4 Configuration

Configuration values are registered in the main `__init__.py` module.

17.4.1 Available Configuration Values

Name	Description	Default
typst_documents	List of documents to build	[]
typst_template	Path to custom template file	None
typst_template_function	Template function name or dict	None
typst_package	Typst Universe package	None
typst_authors	Detailed author information	None
typst_use_mitex	Use mitex for LaTeX math	True
typst_use_codly	Use codly for code highlighting	True
typst_code_line_numbers	Show line numbers in code blocks	True
typst_papersize	Paper size (e.g., “a4”, “us-letter”)	“a4”
typst_fontsize	Base font size	“11pt”

See Configuration for detailed usage of each option.

18 Indices and Tables

- Index
- Module Index

19 Contributing

Thank you for your interest in contributing to typsphinx!

This guide will help you get started with development.

19.1 Development Setup

19.1.1 Prerequisites

- Python 3.9 or later
- uv (recommended) or pip
- Git

19.1.2 Clone and Install

```
# Clone the repository
git clone https://github.com/YuSabo90002/typsphinx.git
cd typsphinx

# Install with development dependencies
uv sync --extra dev

# Or with pip
pip install -e ".[dev]"
```

19.2 Running Tests

We use pytest for testing:

```
# Run all tests
uv run pytest

# Run with coverage
uv run pytest --cov

# Run specific test file
uv run pytest tests/test_builder.py

# Run with verbose output
uv run pytest -v
```

19.2.1 Test Coverage

We maintain 90%+ test coverage. When adding new features:

1. Write tests first (TDD approach)
2. Ensure all tests pass
3. Check coverage doesn't decrease

19.3 Code Quality

We use multiple tools to ensure code quality:

19.3.1 Black (Code Formatting)

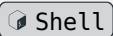
```
# Format all code
uv run black .

# Check without modifying
uv run black --check .
```

19.3.2 Ruff (Linting)

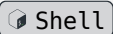
```
# Lint all code
uv run ruff check .

# Auto-fix issues
uv run ruff check --fix .
```



19.3.3 Mypy (Type Checking)


```
# Type check
uv run mypy typsphinx/
```



19.3.4 All Checks

Run all quality checks:

```
uv run black .
uv run ruff check .
uv run mypy typsphinx/
uv run pytest --cov
```



19.3.5 Using Tox

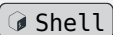
We use tox for running tests across multiple Python versions and environments.

Tox provides the same commands used in CI, making it easy to reproduce issues locally:

```
# Run all tox environments (tests, lint, type check, docs)
uv run tox

# Run specific environments
uv run tox -e lint          # Black + Ruff
uv run tox -e type          # Mypy type checking
uv run tox -e py311         # Tests on Python 3.11
uv run tox -e docs-html     # Build HTML documentation
uv run tox -e docs-pdf      # Build PDF documentation
uv run tox -e docs          # Build both HTML and PDF

# Run tests on specific Python versions
uv run tox -e py39,py310,py311,py312
```




The tox configuration is defined in `tox.ini` and provides:

- Consistent test execution across local and CI environments
- Isolated virtual environments for each test run
- Same commands work locally and in GitHub Actions

19.4 Development Workflow

19.4.1 1. Create a Feature Branch

```
git checkout -b feature/your-feature-name
```



19.4.2 2. Make Changes

- Write code following the project style

- Add tests for new functionality
- Update documentation as needed

19.4.3 3. Run Tests and Checks

```
uv run pytest --cov
uv run black .
uv run ruff check .
uv run mypy typstphinx/
```

Shell

19.4.4 4. Commit Changes

Use conventional commit messages:

```
git commit -m "feat: add new feature"
git commit -m "fix: resolve bug in translator"
git commit -m "docs: update configuration guide"
```

Shell

Commit types:

- feat: New feature
- fix: Bug fix
- docs: Documentation changes
- style: Code style changes (formatting)
- refactor: Code refactoring
- test: Adding tests
- chore: Maintenance tasks

19.4.5 5. Push and Create Pull Request

```
git push origin feature/your-feature-name
```

Shell

Then create a pull request on GitHub.

19.5 Coding Guidelines

19.5.1 Style

- Follow PEP 8 (enforced by Black and Ruff)
- Line length: 88 characters (Black default)
- Use type hints for public APIs
- Write docstrings for public functions/classes

19.5.2 Documentation

Use Google-style docstrings:

```
def convert_node(node: nodes.Node) -> str:
    """Convert a docutils node to Typst markup.

    Args:
        node: The docutils node to convert

    Returns:
        Typst markup string
```

Python

```

Raises:
    ValueError: If node type is unsupported

Example:
>>> node = nodes.paragraph()
>>> convert_node(node)
'#par[...]'
'''
pass

```

19.5.3 Architecture

- **Builder:** Manages build process, file I/O
- **Writer:** Orchestrates document conversion
- **Translator:** Converts individual node types (Visitor pattern)
- **TemplateEngine:** Handles template processing

19.5.4 Testing

- Write unit tests for individual functions
- Write integration tests for complete builds
- Use fixtures for test data
- Test edge cases and error conditions

19.6 Project Structure

```

typsphinx/
├── typsphinx/          # Main package
│   ├── __init__.py     # Extension entry point
│   ├── builder.py      # TypstBuilder
│   ├── pdf.py          # TypstPDFBuilder
│   ├── writer.py       # TypstWriter
│   ├── translator.py   # TypstTranslator
│   ├── template_engine.py # Template processing
│   └── templates/      # Default templates
├── tests/              # Test suite
├── docs/               # Documentation
├── examples/           # Example projects
└── pyproject.toml      # Project configuration

```

19.7 Reporting Issues

When reporting bugs:

1. Check if the issue already exists
2. Provide a minimal reproducible example
3. Include your environment details:
 - Python version
 - Sphinx version
 - typsphinx version
 - Operating system
4. Describe expected vs actual behavior

Use our issue templates on GitHub.

19.8 Feature Requests

For feature requests:

1. Describe the use case
2. Explain why it's needed
3. Suggest implementation approach (optional)
4. Consider creating an OpenSpec proposal for major features

19.9 Community

- **GitHub:** <https://github.com/YuSabo90002/typsphinx>
- **Issues:** <https://github.com/YuSabo90002/typsphinx/issues>
- **Discussions:** Use GitHub Discussions for questions

19.10 Code of Conduct

We follow the Contributor Covenant Code of Conduct:

- Be respectful and inclusive
- Welcome newcomers
- Focus on constructive feedback
- Respect differing viewpoints

19.11 License

By contributing, you agree that your contributions will be licensed under the MIT License.

Thank you for contributing to typsphinx!

20 Changelog

All notable changes to typsphinx are documented here.

The format is based on Keep a Changelog ,
and this project adheres to Semantic Versioning .

For the complete changelog, see CHANGELOG.md in the repository.

20.1 Version 0.4.0 (Current)

Fixed

- Document wrapper (`#{...}`) preservation in nested structures (#61)
- Nested lists generating invalid Typst syntax (#62)
- Unified code mode syntax compliance

Changed

- Implemented stream-based rendering architecture
- Changed `strong()` and `emph()` to use content blocks: `strong({...})`, `emph({...})`
- Updated `link()` format from `link(url)[content]` to `link(url, content)`
- List items now use content blocks with newline separators
- API signatures properly formatted with `+` operator concatenation

20.2 Version 0.3.0

Added

- Documentation site with GitHub Pages deployment
- Comprehensive user guide and examples

- API reference documentation

20.3 Version 0.2.2

Added

- Typst Universe template support (#13)
- Dictionary format for `typst_template_function`
- Detailed author information with `typst_authors`
- charged-ieee template examples

Changed

- Template parameter merging system
- Improved template documentation

20.4 Version 0.2.1

Fixed

- Image file copying in builds
- Path handling for multi-document projects

20.5 Version 0.2.0

Added

- `typstpdf` builder for direct PDF generation
- Self-contained PDF generation with `typst-py`
- Code highlighting with `codly` package
- Math rendering with `mitex`
- Template system with customization support

Changed

- Improved Sphinx integration
- Better error messages
- Enhanced type hints

20.6 Version 0.1.0

Added

- Initial release
- `typst` builder for Typst markup generation
- Basic Sphinx to Typst conversion
- `reStructuredText` support
- Table of contents generation
- Cross-reference support

20.7 Migration Guides

20.7.1 Migrating from 0.2.x to 0.3.x

No breaking changes. Documentation site is a new feature.

20.7.2 Migrating from 0.1.x to 0.2.x

Breaking Changes

None. Version 0.2.0 is backward compatible with 0.1.x.

New Features

- Use `typstpdf` builder for direct PDF generation:

```
# Old way (still works)
sphinx-build -b typst source/ build/typst
typst compile build/typst/index.typ output.pdf

# New way (recommended)
sphinx-build -b typstpdf source/ build/pdf
```

- Configure templates with dict format:

```
# Old way (still works)
typst_template_function = "project"

# New way (more flexible)
typst_template_function = {
    "name": "ieee",
    "params": {
        "abstract": "...",
        "index-terms": ["AI", "ML"],
    }
}
```

20.8 Development Status

- **v0.3.x**: Current stable release
- **v0.2.x**: Maintenance mode
- **v0.1.x**: No longer supported

20.9 Deprecation Policy

We follow semantic versioning:

- **Major versions** (x.0.0): May include breaking changes
- **Minor versions** (0.x.0): New features, backward compatible
- **Patch versions** (0.0.x): Bug fixes, backward compatible

Deprecated features are:

1. Announced in the release notes
2. Kept for at least one minor version
3. Removed in the next major version

20.10 Upcoming Features

See our [GitHub Issues](#)
and [Project Roadmap](#)
for planned features.

20.11 Versioning

typsphinx uses semantic versioning (SemVer):

- **MAJOR**: Incompatible API changes
- **MINOR**: New functionality, backward compatible
- **PATCH**: Bug fixes, backward compatible

20.12 Release Process

1. Update version in `pyproject.toml`

2. Update `CHANGELOG.md`
3. Create git tag: `v0.x.x`
4. Push to GitHub
5. GitHub Actions builds and publishes to PyPI
6. GitHub Release created with changelog

20.13 See Also

- [GitHub Releases](#)
- [PyPI Release History](#)
- [Contributing for development guidelines](#)

21 Indices and tables

- [Index](#)
- [Module Index](#)
- [Search Page](#)