XML to TeX Transformation Engine

A powerful and flexible Node.js-based engine that transforms XML documents into text-based formats (primarily TeX/LaTeX) using a declarative, template-driven approach. This tool excels at converting structured documents like academic papers, technical documentation, and publications from XML to publication-ready TeX format.

Features

- **Template-Driven Transformation**: Output structure follows template design, not source XML structure
- **CSS Selector Matching**: Use familiar CSS-like selectors to target XML elements
- **Processing Instructions Support**: Handle XML processing instructions with custom templates
- **Automatic TeX Escaping**: Built-in character escaping for safe TeX output
- **Smart Whitespace Control**: Configurable whitespace preservation and trimming
- **Flexible Placeholder System**: Multiple placeholder types for content, attributes, and delegation
- **Performance Monitoring**: Built-in performance metrics and memory usage tracking
- **Comprehensive Validation**: Reports on unprocessed XML nodes for debugging

Installation

Prerequisites

- Node.js 14.0.0 or higher
- npm or yarn

Install Dependencies

PROF

```
npm install
```

Global Installation (Optional)

To use the `transform-xml` command globally:

```
npm link
# Now you can use: transform-xml <xml-file> <template-file> [output-
file]
```

Quick Start

1. Prepare your files

Create a simple XML document (`document.xml`):

```
<?xml version="1.0"?>
<article>
    <head>
        <ce:title id="title1">Hello World Example</ce:title>
    </head>
    <body>
        <p id="p1">This is a sample paragraph.</p>
        <ce:italic>emphasized text</ce:italic>
    </body>
</article>
```

Create a template file (`template.xml`):

```
<templates>
    <template data-xml-selector="article">\documentclass{article}
\begin{document}
[[...]]
\end{document}</template>
    <template data-xml-selector="head">[[...]]</template>
    <template data-xml-selector="body">[[...]]</template>
    <template data-xml-selector="ce:title">\title{[[.]]} % ID: [[@id]]
</template>
    <template data-xml-selector="p">\par [[@id]]: [[...]]</template>
    <template data-xml-selector="ce:italic">\textit{[[.]]}</template>
</templates>
```

2. Run the transformation

PROF

```
# Using Node.js directly
node src/cli.js document.xml template.xml output.tex

# Or if globally installed
transform-xml document.xml template.xml output.tex

# With performance monitoring
node src/cli.js document.xml template.xml output.tex --perf

## Template System

The transformation is controlled by a template file containing XML/HTML
rules. All transformation rules must be inside a single `<templates>`
root element.
```

```
### Template Types

#### 1. Basic Template: `<template>`

The primary template type that matches XML elements and defines their
output.

**Required Attribute**: `data-xml-selector` - CSS-like selector to match
XML elements

```xml
<!-- Transform all <p> elements -->
<template data-xml-selector="p">\par [[...]]</template>

<!-- Transform <title> elements with specific attributes -->
<template data-xml-selector="ce:title">\title{[[.]]}</template>
```

2. Null Template: `<null-template>`

Matches elements but produces no output - useful for ignoring unwanted XML sections.

```xml
<!-- Ignore all metadata elements -->
<null-template data-xml-selector="metadata"/>

<!-- Skip processing instructions -->
<null-template data-xml-selector="processing-instruction"/>
```

3. Processing Instruction Template: `<pi-template>`

Handles XML Processing Instructions like `<?tex-break?>` or `<?page-break?>`.

PROF

Required Attribute: `target` - The PI target name
Optional Attribute: `match` - Additional matching criteria

```xml
<!-- Handle <?tex-kern amount="5pt"?> -->
<pi-template target="tex-kern">\kern [[@amount]]</pi-template>

<!-- More specific matching -->
<pi-template target="spacing"
match="type='vertical'">\vspace{[[@amount]]}</pi-template>
```

4. Unprocessed Template: `<unprocessed-template>`

Defines fallback output for XML elements that don't match any other template.
```

```
<unprocessed-template>
    \fbox{UNHANDLED: [[@tagName]] - [[.]]}
</unprocessed-template>
```

CSS Selector Support

The engine supports a subset of CSS selectors for matching XML elements:

Selector Type	Example	Description
Element	`p`	Matches all `<p>` elements
Universal	`*`	Matches all elements
Attribute (exists)	`[label]`	Elements with a `label` attribute
Attribute (value)	`[type="note"]`	Elements where `type="note"`
Descendant	`article p`	`<p>` elements anywhere inside `<article>`
Child	`head > title`	`<title>` elements that are direct children of `<head>`
Namespace	`ce:title`	Elements with namespace prefix
Complex	`section[id] > p`	Direct `<p>` children of `<section>` elements with `id`

Selector Specificity: When multiple templates match the same element, CSS specificity rules determine which template to use:

- Attribute selectors have higher specificity than element selectors
- More specific selectors (e.g., `div[class="warning"]`) override less specific ones (e.g., `div`)
- Child combinators (`>`) are more specific than descendant combinators (space)

Placeholder System

PROF

Placeholders are special commands inside templates that extract data from XML elements. All placeholders use `[[...]]` delimiters.

Core Placeholders

Placeholder	Description	Example Usage
`[[...]]`	**Delegate** - Process all child nodes	`<template data-xml-selector="body"> [[...]]</template>`
`[[.]]`	**Text Content** - Extract text content only	`<template data-xml-selector="title">\title{[[.]]} </template>`

Placeholder	Description	Example Usage
`[[@attr]]`	**Attribute** - Extract attribute value	`<template data-xml-selector="p">\paragraph{[[@id]]}</template>`
`[[@tagName]]`	**Tag Name** - Extract element tag name	`\command{[[@tagName]]}{[[.]]}`

Advanced Placeholders

Placeholder	Description	Context
`[[@target]]`	PI target name	Processing Instructions only
`[[@data]]`	PI data content	Processing Instructions only
`[[selector:...]]`	**Scoped** - Apply to selected child elements	`[[ce:title:.]]`

Filters

Placeholders support filter pipelines using the `|` operator:

Filter	Description	Example
`raw`	Skip TeX escaping	`[[@id | raw]]`
Custom filters	User-defined transformations	`[[. | uppercase]]`

Practical Examples

```
<!-- XML Input -->
<section id="intro" type="chapter">
    <title>Introduction</title>
    <p>Welcome to the guide.</p>
    <?page-break type="soft"?>
</section>

<!-- Template -->
<templates>
    <template data-xml-selector="section">\section{[[@id | raw]]}
\label{sec:[[@id]]}
[[...]]
    </template>

    <template data-xml-selector="title">\subsection{[[.]]}</template>

    <template data-xml-selector="p">[[.]]\par</template>

    <pi-template target="page-break">\newpage</pi-template>
</templates>
```

```
<!-- Output -->
\section{intro}
\label{sec:intro}
\subsection{Introduction}
Welcome to the guide.\par
\newpage
```

Advanced Features

Template Composition

Use `<apply-template>` and `<apply-children>` for modular template design:

```
<template data-xml-selector="article" xml:space="preserve">
\documentclass[12pt]{article}
\begin{document}
    <apply-template data-xml-selector="head"/>
    <apply-template data-xml-selector="body"/>
\end{document}
</template>
```

Whitespace Control

By default, templates use smart whitespace trimming. Use `xml:space="preserve"` to maintain exact formatting:

```
<template data-xml-selector="code" xml:space="preserve">
\begin{verbatim}
[[.]]
\end{verbatim}
</template>
```


TeX Character Escaping

The engine automatically escapes TeX special characters:

- `&` → `\&`
- `%` → `\%`
- `$` → `\$`
- `#` → `\#`
- `_` → `\_`
- `{` → `\{`
- `}` → `\}`
- `\` → `\textbackslash{}`

- `^` → `\textasciicircum{}`
- `~` → `\textasciitilde{}`

Use the `| raw` filter to skip escaping when needed:

```
<template data-xml-selector="math">$[[@formula | raw]]$</template>
```

CLI Reference

Basic Usage

```
node src/cli.js <xml-file> <template-file> [output-file] [options]
```

Options

Option	Description
`--perf`	Display performance metrics (memory, CPU usage)
`--run-test`	Run built-in test suite

Examples

```
# Basic transformation
node src/cli.js template/example.xml template/main.tex.xml output.tex

# Output to stdout
node src/cli.js template/example.xml template/main.tex.xml

# With performance monitoring
node src/cli.js template/example.xml template/main.tex.xml output.tex --
perf

# Run tests
node src/cli.js --run-test
```

PROF

Library Usage

Use the transformation engine programmatically:

```
const { transform } = require('./src/engine.js');

async function example() {
    const xmlString = `<article><title>Hello</title></article>`;
```

```
    const templateString = `
        <templates>
            <template data-xml-selector="article">[[...]]</template>
            <template data-xml-selector="title">\\title{[[.]]}
</template>
        </templates>
    `;

    const result = await transform(xmlString, templateString);

    console.log('Output:', result.output);
    console.log('Unprocessed nodes:',
result.report.unprocessedNodes.length);
    console.log('Memory used:', result.performance.memoryUsage.heapUsed,
'MB');
}
```

Custom Processors

```
const customProcessors = {
    filters: {
        uppercase: (text) => text.toUpperCase(),
        prefix: (text, node, context) => `PREFIX: ${text}`
    }
};

const context = {
    engine: {
        escapeFn: (text) => text.replace(/&/g, '\\&') // Custom escaping
    }
};

const result = await transform(xmlString, templateString,
customProcessors, context);
```

PROF

Testing

Built-in Tests

Run the internal test suite:

```
node src/cli.js --run-test
```

Manual Testing

Test with the provided example template:

```
# Create a test XML file
echo '<?xml version="1.0"?>
<article>
    <head><ce:title id="t1">Sample Title</ce:title></head>
    <body><p id="p1">Test paragraph</p></body>
</article>' > test.xml

# Run transformation
node src/cli.js test.xml template/main.tex.xml output.tex
```

Unit Testing

Run the TeX engine-specific tests:

```
node src/test-tex-engine.js
```

Project Structure

```
xml2tex/
├── src/
│   ├── cli.js              # Command-line interface
│   ├── engine.js           # Core transformation engine
│   └── test-tex-engine.js  # Unit tests
├── template/
│   └── main.tex.xml        # Example template
├── package.json            # Project configuration
├── README.md              # This file
└── .gitignore            # Git ignore rules
```

PROF

Dependencies

- **nanoid**: Unique ID generation for XML elements
- **peggy**: PEG parser generator for CSS selectors and placeholders
- **xmldom**: XML parsing and manipulation

Performance

The engine includes built-in performance monitoring:

- Memory usage tracking (heap allocation)
- CPU usage measurement (user/system time)
- Processing time measurement
- Unprocessed node reporting for optimization

Use the `--perf` flag to display detailed metrics:

```
node src/cli.js document.xml template.xml output.tex --perf
```

Troubleshooting

Common Issues

1. **"Unprocessed nodes" warnings**: Add templates for missing XML elements or use `<null-template>` to ignore them
2. **Template not matching**: Check CSS selector syntax and XML namespace prefixes
3. **TeX compilation errors**: Verify that special characters are properly escaped or use the `| raw` filter appropriately
4. **Performance issues**: Use performance monitoring to identify bottlenecks in large documents

Debug Mode

The engine reports unprocessed nodes by default. To get detailed information:

```
const result = await transform(xmlString, templateString);
console.log('Unprocessed nodes:', result.report.unprocessedNodes);
```

Contributing

1. Fork the repository
2. Create a feature branch: `git checkout -b feature/new-feature`
3. Make changes and add tests
4. Run the test suite: `node src/cli.js --run-test`
5. Commit changes: `git commit -am 'Add new feature'`
6. Push to branch: `git push origin feature/new-feature`
7. Submit a Pull Request

PROF

Development Setup

```
# Clone the repository
git clone <repository-url>
cd xml2tex

# Install dependencies
npm install

# Run tests
node src/cli.js --run-test
node src/test-tex-engine.js
```

```
# Test CLI
node src/cli.js --help
```

License

ISC License - see package.json for details.

Changelog

v1.0.0

- Initial release
- Template-driven XML to TeX transformation
- CSS selector support for element matching
- Processing instruction handling
- Automatic TeX character escaping
- Performance monitoring
- Comprehensive placeholder system