SmoovOps Template Syntax Reference

Generated on 2026-08-08

Template Syntax Reference

This page documents the full syntax of the restricted, Pug-style language used to write a document template's **Header**, **Content**, and **Footer** in Managing document templates. It's aimed at anyone on the **Pro** plan authoring their own templates — an account Owner or Admin, or someone they've handed the job to. A standalone PDF of this page is also available, so it can be handed to a template author who has no need for the rest of the manual.

This is a deliberately small **subset** of the real Pug language — several things a Pug-familiar reader might reach for out of habit aren't supported here, on purpose (see What's not supported below). Every rule on this page is exactly what SmoovOps's template compiler accepts; if you paste something from elsewhere and it doesn't compile, this page — together with the exact error message you'll see — is the place to check why.

Basic structure

A template is a plain text file. Nesting is expressed through indentation — **spaces only, no tabs** — the same way Pug always works: there are no closing tags. The first time a line indents further than the one before it, that indent width becomes fixed for the rest of the template; every later indent must use the same width, or the template fails to compile.

The only supported doctype line is:

```
doctype html
```

which compiles to `<!doctype html>`. Nothing else is accepted on a `doctype` line.

Tags

A tag line is a tag name, optionally followed by `.class` and `#id` shorthand, optionally followed by attributes in parentheses, optionally followed by a space and inline text:

```
div.card#summary(data-role="note") Hello there
```

compiles to:

```
<div class="card" id="summary" data-role="note">Hello there</div>
```

- The tag name may contain only letters and digits (no hyphens — custom element names like `my-widget` aren't supported).
- You can chain as many `.class` and `#id` pieces as you like, in any order.
- If you omit the tag name entirely and start with `.` or `#`, the tag defaults to `div` — `.card` alone is the same as `div.card`.
- Instead of inline text, a tag can have nested child tags on the following, more-indented lines:

```
.document
  p Line one
  p Line two
```

- **Void elements** (`img`, `br`, `hr`, `input`, `meta`, `link`, and the other HTML elements that never take content) never take inline text or children, matching normal HTML.

Attributes

Attributes go in parentheses after the tag head, comma-separated, and **must be quoted strings**:

```
img(src="/logo.png", alt="Company logo")
```

An attribute value may contain one or more `#{...}` interpolations (see below), but it cannot be a bare, unquoted expression. This is invalid and fails to compile:

```
a(href=recipient.url) Visit
```

with the error:

```
dynamic attribute values are not supported near "href=recipient.url" — use a quoted
string, optionally with #{field.path}
```

Interpolation: `#{...}`

Inside tag text or an attribute value, `#{field.path}` inserts a value from the data the document is generated with. You can use as many interpolations as you like on one line, mixed with literal text:

```
p #{recipient.name}, #{recipient.street}, #{recipient.postalCode} #{recipient.city}
```

The path inside `#{...}` **must be a plain dot-path** — a field name, or a chain of field names separated by dots, like `recipient.name` or `sender.street`. Anything else — arithmetic, function calls, comparisons, string concatenation — is rejected:

```
p #{price * qty}
```

```
"price * qty" in tag text is not a plain field path — only dot-paths like "recipient.name"
are supported; compute values before generating the document and source them as an
already-computed field
```

In practice this means: if you need a computed value, it has to already exist as one of the fields you declare on the template (see What data is available below) — templates can display data, not calculate it.

Loops: `each ... in ...`

```
table
  each item in items
    tr
      td #{item.description}
      td #{item.quantity}
```

`each <name> in <path>` iterates the array found at `<path>` (also a plain dot-path — the same rule as `#{...}` applies). Inside the loop body, refer to each item's fields through the loop variable exactly as you named it — `#{item.description}`, `#{item.quantity}`, and so on, whatever name you chose after `each`.

- Only a single loop variable is supported — there's no destructuring (`each {a, b} in ...`) and no index variable (`each item, i in ...`). Writing either produces:

```
malformed "each" — destructuring and index variables are not supported; use each item in
items
```

- Loops can be nested, and can contain any other construct (tags, `if`, further nested `each`).

Conditionals: `if ...`

```
if item.taxable
  td.taxed Taxed
```

`if <path>` shows its block only when the value at that dot-path is truthy — anything except missing, `null`, `false`, `0`, or an empty string. There is **no `else`**, and no comparison or negation operators (`==`, `!=`, `!`, `&&`, `||`) — only a bare dot-path, exactly like `#{...}` and `each`. If you need an either/or, declare two fields and set exactly one of them, or use two separate `if` blocks each checking its own field.

Raw text blocks: `tag.`

A tag whose head ends in a bare `.` with nothing after it becomes a **block-text** tag: every line indented further than it is copied into the output exactly as written — no `#{...}` interpolation, no escaping. This exists for things like a `<style>` block of static CSS:

```
style.
  @page { margin: 0; size: A4; }
  body { font-family: Arial, Helvetica, sans-serif; }
```

Because block-text content is never interpolation-processed, `#{...}` inside one is left as literal text, not resolved — don't rely on it there. If a style rule genuinely needs to vary per document (for example, a page size that depends on language), hardcode the value you need for that template rather than trying to interpolate it inside `style.`.

Comments: `//-`

A line starting with `//-` is a comment and is stripped entirely — it produces no output at all:

```
//- This line explains something to the next editor and won't appear in the document.
p Visible text
```

Plain Pug also supports a *buffered* comment (`//`, without the trailing `-`) that gets kept in the rendered HTML — that form isn't supported here; only the stripped `//-` form is.

What's not supported

These constructs exist in full Pug but are rejected by this restricted subset, each with its own error message so you know immediately what to remove:

Construct	Example	Error
`mixin` / `extends` / `include` / `block` / `case` / `when` / `while`	`mixin card`	`"mixin" is not supported in this restricted Pug subset`
Mixin call	`+card`	`mixin calls (+name) are not supported in this restricted Pug subset`
Buffered code	`= 1 + 1`	`buffered code (=) is not supported in this restricted Pug subset`
Unescaped buffered code	`!= raw`	`unescaped buffered code (!=) is not supported in this restricted Pug subset`
Unbuffered code	`- const x = 1`	`unbuffered code (-) is not supported in this restricted Pug subset`
Filters	`:markdown`	`filters (:filter) are not supported in this restricted Pug subset`

Content safety

A few constructs are rejected outright, not because the *syntax* is unsupported, but because they'd let a template author embed something dangerous into a document other people in your account (or a partner reading it) might later see:

- A `<script>` tag anywhere: `the "<script>" tag is not allowed in templates`.
- Any event-handler attribute (`onclick`, `onload`, and so on): `event-handler attribute "onclick" is not allowed in templates`.
- An attribute value that's a literal `javascript:` URI: `attribute "href" uses a "javascript:" URI, which is not allowed in templates`.

This is separate from — and in addition to — the fact that every `#{...}` value is always HTML-escaped at render time no matter what the template looks like, so data typed into a generated document (an address, an "Additional text" field, and so on) can never inject markup either. Both protections exist together: this one guards against a careless template *author*, the escaping guards against the *data* a template is later filled in with.

What data is available

What you can reference with `#{...}`, `each`, and `if` depends on the document being generated:

- **`sender.*`** — your company details, from Settings → Company Profile and its **Main** address. Includes `sender.name`, `sender.street`, `sender.zip`, `sender.city`, and — when set — `sender.country`, `sender.contactName`, `sender.email`, `sender.phone`, `sender.taxId` / `sender.vatId` / `sender.registrationNumber`, `sender.website`, and `sender.logoDataUri` (a data URI, ready to drop straight into an `img` tag's `src`). A Letter, Offer, or Invoice template can't be generated at all until a Company Profile with at least a Legal name exists — see Managing your company profile — so unlike the other fields on this page, `sender.*` is never empty when your template actually runs against real data.
- **`recipient.*`** — filled automatically from the selected Contract's partner when one is picked in Generate Document, otherwise typed in by hand via the fields you declare (see below). Typically `recipient.name`, `recipient.street`, `recipient.postalCode`, `recipient.city`.
- **`date`** — the current date, already formatted for the app's language.
- **`additionalText`** — the free-text box every generated document has. It gets special handling: line breaks the user typed become `<br>` in the output (after escaping, so this never turns into a real tag by accident) — everywhere else, a value is inserted as plain text.
- **`items`** (Offer/Invoice templates only) — the line items added in Generate Document, for use with `each item in items`; each item has its own fields such as `description`, `quantity`, `unitPrice`, and `total`. **`taxBreakdown`** is a second array, one entry per tax rate in use, for a per-rate subtotal table.
- **Any field you declare yourself** — every row you add in the template editor's field list (name, label, type, required) becomes available under the exact dot-path you gave it as its **Name**. A field named `dueDate` is referenced as `#{dueDate}`; a field named `invoice.number` is referenced as `#{invoice.number}`.

A path that isn't supplied at render time and isn't declared as a field simply renders as nothing — it's never an error.

Two complete examples

Both of the examples below were compiled and rendered against sample data before being written into this page, so what you see is exactly what the compiler produces — not a hand-typed approximation.

A minimal letter

```
doctype html
html
  head
    meta(charset="utf-8")
    title #{subject}
  body
    .document
      .recipient
        p #{recipient.name}
        p #{recipient.street}
        p #{recipient.postalCode} #{recipient.city}
      p.subject #{subject}
      p #{salutation}
      p #{additionalText}
      p #{closing}
      p #{sender.name}
```

Filled in with a recipient, a subject, and some additional text, this renders to a normal HTML letter body — no surprises, no left-over `{{...}}` or `#{...}` markers.

An invoice-style item table with a conditional

```
table.items
  thead
    tr
      th Description
      th Qty
      th Tax
  tbody
    each item in items
      tr(class="item-row")
        td #{item.description}
        td #{item.quantity}
        if item.taxable
          td.taxed Taxed
```

Rendered against two items — one taxable, one not — the taxable row gets an extra "Taxed" cell and a `taxed` class; the other row simply doesn't, since there's no `else` to fall back to (see Conditionals above).

Tips

- **Validate early and often.** The Header, Content, and Footer sections are compiled and checked independently — an error in one is labeled with which section it came from — and a successful validation shows a live preview against realistic sample data, including your company logo if you've turned it on.
- **Keep Header and Footer content compact.** When a header or footer's visibility is set to "All pages," it's reserved a fixed vertical band at the top or bottom of every printed page (72px for a header, 56px for a footer) — content that doesn't fit in that space will overlap the page content around it.
- **Save stays disabled until you've validated the exact text you're about to save.** Editing any of the three sections after a successful Validate disables Save again, so you can never publish a template that hasn't actually been checked.