SmoovOps Vorlagen-Syntax-Referenz

Erstellt am 2026-08-08

Vorlagen-Syntax-Referenz

Diese Seite dokumentiert die vollständige Syntax der eingeschränkten, an Pug angelehnten Sprache, mit der du **Kopfzeile**, **Inhalt** und **Fußzeile** einer Dokumentvorlage schreibst — siehe Dokumentvorlagen verwalten. Sie richtet sich an alle im **Pro**-Tarif, die eigene Vorlagen erstellen — einen Konto-Inhaber oder Admin, oder jemanden, dem diese Aufgabe übertragen wurde. Eine eigenständige PDF dieser Seite ist ebenfalls verfügbar, sodass sie an eine Person weitergegeben werden kann, die den Rest des Handbuchs nicht braucht.

Dies ist bewusst eine kleine **Teilmenge** der echten Pug-Sprache — einige Dinge, die eine mit Pug vertraute Person aus Gewohnheit verwenden würde, werden hier absichtlich nicht unterstützt (siehe Was nicht unterstützt wird unten). Jede Regel auf dieser Seite entspricht genau dem, was der Vorlagen-Compiler von SmoovOps akzeptiert; falls du etwas von anderswo einfügst und es nicht kompiliert, findest du hier — zusammen mit der genauen Fehlermeldung, die du siehst — den Grund.

Grundstruktur

Eine Vorlage ist eine reine Textdatei. Verschachtelung wird durch Einrückung ausgedrückt — **nur Leerzeichen, keine Tabs** — genau wie bei Pug üblich: Es gibt keine schließenden Tags. Sobald eine Zeile das erste Mal weiter eingerückt ist als die vorherige, ist diese Einrücktiefe für den Rest der Vorlage festgelegt; jede spätere Einrückung muss dieselbe Tiefe verwenden, sonst schlägt die Kompilierung fehl.

Die einzige unterstützte Doctype-Zeile ist:

```
doctype html
```

was zu `<!doctype html>` kompiliert. Alles andere in einer `doctype`-Zeile wird nicht akzeptiert.

Tags

Eine Tag-Zeile besteht aus einem Tag-Namen, optional gefolgt von `.class`- und `#id`-Kurzform, optional gefolgt von Attributen in Klammern, optional gefolgt von einem Leerzeichen und Inline-Text:

```
div.card#summary(data-role="note") Hallo zusammen
```

kompiliert zu:

```
<div class="card" id="summary" data-role="note">Hallo zusammen</div>
```

- Der Tag-Name darf nur Buchstaben und Ziffern enthalten (keine Bindestriche — eigene Element-Namen wie `my-widget` werden nicht unterstützt).
- Du kannst beliebig viele `.class`- und `#id`-Teile in beliebiger Reihenfolge aneinanderreihen.
- Lässt du den Tag-Namen ganz weg und beginnst mit `.` oder `#`, ist der Tag standardmäßig ein `div` — `.card` allein ist dasselbe wie `div.card`.
- Statt Inline-Text kann ein Tag verschachtelte Kind-Tags in den folgenden, weiter eingerückten Zeilen haben:

```
.document
  p Erste Zeile
  p Zweite Zeile
```

- **Void-Elemente** (`img`, `br`, `hr`, `input`, `meta`, `link` und die anderen HTML-Elemente, die nie Inhalt haben) nehmen — wie im normalen HTML — nie Inline-Text oder Kind-Tags auf.

Attribute

Attribute stehen in Klammern nach dem Tag-Kopf, durch Kommas getrennt, und **müssen als Zeichenketten in Anführungszeichen** stehen:

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

Ein Attributwert kann ein oder mehrere `#{...}`-Interpolationen enthalten (siehe unten), darf aber kein bloßer, unquotierter Ausdruck sein. Dies ist ungültig und kompiliert nicht:

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

mit der Fehlermeldung:

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

Interpolation: `#{...}`

Innerhalb von Tag-Text oder eines Attributwerts fügt `#{feld.pfad}` einen Wert aus den Daten ein, mit denen das Dokument erzeugt wird. Du kannst beliebig viele Interpolationen in einer Zeile verwenden, gemischt mit reinem Text:

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

Der Pfad innerhalb von `#{...}` **muss ein reiner Punktpfad** sein — ein Feldname oder eine durch Punkte getrennte Kette von Feldnamen, wie `recipient.name` oder `sender.street`. Alles andere — Rechenoperationen, Funktionsaufrufe, Vergleiche, String-Verkettung — wird abgelehnt:

```
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 der Praxis bedeutet das: Brauchst du einen berechneten Wert, muss er bereits als eines der Felder existieren, die du in der Vorlage deklarierst (siehe Welche Daten verfügbar sind unten) — Vorlagen zeigen Daten an, sie berechnen sie nicht.

Schleifen: `each ... in ...`

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

`each <name> in <pfad>` iteriert über das Array am angegebenen `<pfad>` (ebenfalls ein reiner Punktpfad — dieselbe Regel wie bei `#{...}`). Innerhalb des Schleifenkörpers greifst du auf die Felder jedes Elements über die Schleifenvariable zu, genau so, wie du sie benannt hast — `#{item.description}`, `#{item.quantity}` usw., je nachdem, welchen Namen du nach `each` gewählt hast.

- Es wird nur eine einzelne Schleifenvariable unterstützt — es gibt kein Destructuring (`each {a, b} in ...`) und keine Index-Variable (`each item, i in ...`). Beides führt zu:

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

- Schleifen können verschachtelt werden und beliebige andere Konstrukte enthalten (Tags, `if`, weitere verschachtelte `each`).

Bedingungen: `if ...`

```
if item.taxable
  td.taxed Besteuert
```

`if <pfad>` zeigt seinen Block nur, wenn der Wert an diesem Punktpfad wahr ist — also alles außer fehlend, `null`, `false`, `0` oder ein leerer String. Es gibt **kein `else`** und keine Vergleichs- oder Verneinungsoperatoren (`==`, `!=`, `!`, `&&`, `||`) — nur einen reinen Punktpfad, genau wie bei `#{...}` und `each`. Brauchst du ein Entweder-Oder, deklariere zwei Felder und setze genau eines davon, oder verwende zwei getrennte `if`-Blöcke, die jeweils ihr eigenes Feld prüfen.

Rohtext-Blöcke: `tag.`

Ein Tag, dessen Kopf mit einem alleinstehenden `.` endet, wird zu einem **Rohtext**-Tag: Jede Zeile, die weiter eingerückt ist als er, wird unverändert in die Ausgabe übernommen — keine `#{...}`-Interpolation, kein Escaping. Das ist für Dinge wie einen `<style>`-Block mit statischem CSS gedacht:

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

Da Rohtext-Inhalt nie interpolationsverarbeitet wird, bleibt `#{...}` darin als reiner Text stehen, statt aufgelöst zu werden — verlass dich dort nicht darauf. Muss eine Style-Regel tatsächlich je Dokument variieren (zum Beispiel eine Seitengröße abhängig von der Sprache), schreibe den benötigten Wert für diese Vorlage fest, statt zu versuchen, ihn innerhalb von `style.` zu interpolieren.

Kommentare: `//-`

Eine Zeile, die mit `//-` beginnt, ist ein Kommentar und wird vollständig entfernt — sie erzeugt keinerlei Ausgabe:

```
//- Diese Zeile erklärt der nächsten Person etwas und erscheint nicht im Dokument.
p Sichtbarer Text
```

Reines Pug kennt zusätzlich einen *gepufferten* Kommentar (`//`, ohne den abschließenden `-`), der im gerenderten HTML erhalten bleibt — diese Form wird hier nicht unterstützt; nur die entfernte `//-`-Form.

Was nicht unterstützt wird

Diese Konstrukte gibt es im vollständigen Pug, werden aber von dieser eingeschränkten Teilmenge abgelehnt, jeweils mit eigener Fehlermeldung, damit du sofort weißt, was zu entfernen ist:

Konstrukt	Beispiel	Fehler
`mixin` / `extends` / `include` / `block` / `case` / `when` / `while`	`mixin card`	`"mixin" is not supported in this restricted Pug subset`
Mixin-Aufruf	`+card`	`mixin calls (+name) are not supported in this restricted Pug subset`
Gepufferter Code	`= 1 + 1`	`buffered code (=) is not supported in this restricted Pug subset`
Ungeschützter gepufferter Code	`!= raw`	`unescaped buffered code (!=) is not supported in this restricted Pug subset`
Ungepufferter Code	`- const x = 1`	`unbuffered code (-) is not supported in this restricted Pug subset`
Filter	`:markdown`	`filters (:filter) are not supported in this restricted Pug subset`

Inhaltssicherheit

Einige Konstrukte werden grundsätzlich abgelehnt — nicht weil ihre *Syntax* unzulässig wäre, sondern weil sie es einer Vorlagenautorin oder einem -autor erlauben würden, etwas Gefährliches in ein Dokument einzubetten, das andere Personen in deinem Konto (oder ein Partner, der es liest) später sehen könnten:

- Ein `<script>`-Tag an beliebiger Stelle: `the "<script>" tag is not allowed in templates`.
- Jedes Event-Handler-Attribut (`onclick`, `onload` usw.): `event-handler attribute "onclick" is not allowed in templates`.
- Ein Attributwert, der eine wörtliche `javascript:`-URI ist: `attribute "href" uses a "javascript:" URI, which is not allowed in templates`.

Das ist getrennt von — und zusätzlich zu — der Tatsache, dass jeder `#{...}`-Wert beim Rendern immer HTML-escaped wird, unabhängig davon, wie die Vorlage aussieht, sodass in ein erzeugtes Dokument eingegebene Daten (eine Adresse, ein Feld „Zusätzlicher Text" usw.) ebenfalls niemals Markup einschleusen können. Beide Schutzmechanismen bestehen nebeneinander: Dieser hier schützt vor einer unachtsamen Vorlagen*autorin*, das Escaping schützt vor den *Daten*, mit denen eine Vorlage später ausgefüllt wird.

Welche Daten verfügbar sind

Was du mit `#{...}`, `each` und `if` referenzieren kannst, hängt vom erzeugten Dokument ab:

- **`sender.*`** — deine Firmendaten, aus Einstellungen → Firmenprofil und dessen **Haupt**-Adresse. Enthält `sender.name`, `sender.street`, `sender.zip`, `sender.city` und — falls gesetzt — `sender.country`, `sender.contactName`, `sender.email`, `sender.phone`, `sender.taxId` / `sender.vatId` / `sender.registrationNumber`, `sender.website` sowie `sender.logoDataUri` (eine Data-URI, direkt einsetzbar als `src` eines `img`-Tags). Ein Brief, Angebot oder eine Rechnung kann erst erzeugt werden, wenn ein Firmenprofil mit mindestens einem Rechtlichen Namen existiert — siehe Firmenprofil verwalten — daher ist `sender.*` anders als die übrigen Felder auf dieser Seite nie leer, wenn deine Vorlage tatsächlich mit echten Daten läuft.
- **`recipient.*`** — wird automatisch aus dem Partner des ausgewählten Vertrags übernommen, sobald einer in „Dokument erzeugen" ausgewählt wird, sonst von Hand über die von dir deklarierten Felder eingegeben (siehe unten). Typischerweise `recipient.name`, `recipient.street`, `recipient.postalCode`, `recipient.city`.
- **`date`** — das aktuelle Datum, bereits für die Sprache der App formatiert.
- **`additionalText`** — das Freitextfeld, das jedes erzeugte Dokument hat. Es wird speziell behandelt: Zeilenumbrüche, die eingegeben wurden, werden in der Ausgabe zu `<br>` (nach dem Escaping, damit daraus nie versehentlich ein echtes Tag wird) — überall sonst wird ein Wert als reiner Text eingefügt.
- **`items`** (nur Angebots-/Rechnungsvorlagen) — die in „Dokument erzeugen" hinzugefügten Positionen, zur Verwendung mit `each item in items`; jede Position hat eigene Felder wie `description`, `quantity`, `unitPrice` und `total`. **`taxBreakdown`** ist ein zweites Array, ein Eintrag pro verwendetem Steuersatz, für eine Zwischensumme je Satz.
- **Jedes selbst deklarierte Feld** — jede Zeile, die du in der Feldliste des Vorlagen-Editors hinzufügst (Name, Bezeichnung, Typ, Pflichtfeld), wird unter genau dem Punktpfad verfügbar, den du ihr als **Name** gegeben hast. Ein Feld namens `dueDate` wird als `#{dueDate}` referenziert; ein Feld namens `invoice.number` als `#{invoice.number}`.

Ein Pfad, der beim Rendern nicht mitgeliefert und nicht als Feld deklariert wurde, rendert einfach zu nichts — das ist nie ein Fehler.

Zwei vollständige Beispiele

Beide folgenden Beispiele wurden kompiliert und gegen Beispieldaten gerendert, bevor sie auf diese Seite geschrieben wurden — was du siehst, ist also genau das, was der Compiler erzeugt, keine handgetippte Annäherung.

Ein minimaler Brief

```
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}
```

Mit Empfänger, Betreff und etwas zusätzlichem Text ausgefüllt, ergibt das einen normalen HTML-Briefkörper — keine Überraschungen, kein übrig gebliebenes `{{...}}` oder `#{...}`.

Eine Rechnungs-Artikeltabelle mit einer Bedingung

```
table.items
  thead
    tr
      th Beschreibung
      th Menge
      th Steuer
  tbody
    each item in items
      tr(class="item-row")
        td #{item.description}
        td #{item.quantity}
        if item.taxable
          td.taxed Besteuert
```

Gegen zwei Positionen gerendert — eine steuerpflichtig, eine nicht — erhält die steuerpflichtige Zeile eine zusätzliche „Besteuert"-Zelle und die Klasse `taxed`; die andere Zeile bekommt sie einfach nicht, da es kein `else` gibt, auf das zurückgefallen werden könnte (siehe Bedingungen oben).

Tipps

- **Validiere früh und oft.** Kopfzeile, Inhalt und Fußzeile werden unabhängig voneinander kompiliert und geprüft — ein Fehler in einem von ihnen ist mit dem Abschnitt gekennzeichnet, aus dem er stammt — und eine erfolgreiche Validierung zeigt eine Live-Vorschau mit realistischen Beispieldaten, einschließlich deines Firmenlogos, falls aktiviert.
- **Halte Kopf- und Fußzeile kompakt.** Ist die Sichtbarkeit einer Kopf- oder Fußzeile auf „Alle Seiten" gesetzt, wird oben oder unten auf jeder gedruckten Seite ein fester Bereich reserviert (72px für eine Kopfzeile, 56px für eine Fußzeile) — Inhalt, der dort nicht hineinpasst, überlappt mit dem umgebenden Seiteninhalt.
- **Speichern bleibt deaktiviert, bis genau der Text validiert wurde, den du speichern möchtest.** Änderst du einen der drei Abschnitte nach einer erfolgreichen Validierung, deaktiviert sich Speichern erneut — so kannst du nie eine Vorlage veröffentlichen, die tatsächlich nicht geprüft wurde.