Das 'sopra-requirements'-Paket

Dokumentation für das 'sopra-requirements'-Paket | Version v1.0.09
2. Dezember 2019
Florian Sihler (florian.sihler@uni-ulm.de)

1 Allgemeines

1.1 Warum, wieso, weshalb?

Dieses LaTeX 2_ε-Paket wurde im Rahmen des Sopras im Wintersemester 2019 und Sommersemester 2020 verfasst und dient als Grundlage für die Definition von Anforderungen (funktional, qualitativ, ...) des *Teams 20*. Diese Dokumentation wurde zusammen mit der `sopra-base.cls` sowie dem Paket `sopra-documentation.sty` kreiert.

Zum Visualisieren der einzelnen Code-Ausschnitte wird das `sopra-listings`-Paket verwendet. Das zugehörige Paket sollte ebenfalls in dieses Dokument eingebettet sein: .

1.2 Abhängigkeiten

Dieses Paket bindet die folgenden Paketen mit ein:

- `tabu` (usetabu)
- `booktabs` (notabu)
- `colortbl` (usetabu)
- `makecell`
- `longtable`
- `xcolor`
- `environ`
- `amssymb`
- `etoolbox`
- `fontawesome`
- `hyperref`

All diese Pakete sollten Teil der gängigen LaTeX-Distribution sein.

1.3 Die Installation

Das Paket wird nicht als `.dtx` ausgeliefert, weswegen sich die folgenden Möglichkeiten ergeben:

- Das Paket kann in dasselbe Verzeichnis wie das Dokument gesetzt werden. In diesem Fall lautet die Einbindungsanweisung:

  `\usepackage{sopra-requirements}`

- Das Paket kann in ein Unterverzeichnis/in ein mit dem Dokument ausgeliefertes Verzeichnis gelegt werden. In diesem Fall erfolgt die Angabe durch den (relativen-) Pfad:

  `\usepackage{./Mein/Pfad/zu/sopra-requirements}`

- Man kann das Paket (mittels eines Symlinks oder ähnlichem) in einen eigenen *texmf*-Baum ablegen. So kann zum Beispiel auf Linux unter der Verwendung von texlive das Paket hier abgelegt werden: `~/texmf/tex/latex/`. Das Verzeichnis kann erstellt und anschließend mittels `texhash ~/texmf` aktualisiert werden. Nun kann das Paket wie jede andere installierte Paket verwendet werden:

  `\usepackage{sopra-requirements}`

1.4 Weitere Besonderheiten

In Version v1.0.09 (`\thesorversion` → p. 3) gibt es keine weiteren Besonderheiten.

2 Paket-Konfiguration

2.1 Akzeptierte Parameter

▷ usetabu (notabu)

Standardmäßig verwendet dieses Paket aus optischen Gründen tabu um die Tabellen zu visualisieren. Sollten die in diesem paket enthaltenen (bug-)fixes (nicht mehr) funktionieren, oder einem die in notabu verwendeten booktabs Tabellen einem bessere gefallen, so kann diese Option entsprechend umgestellt werden.

▷ intoc (notoc)

Standardmäßig fügt das tabu-Paket die Requirements in den table of contents mit ein (als subsubsection, wird also in der Standardmäßig nur im TOC des PDF-Viewers angezeigt). Wenn dies nicht gewünscht ist, kann man es durch notoc deaktivieren.

▷ showtag (noshowtag)

Zeigt nebst der ID auch noch den (internen) Bezeichner der Anforderung an. Hierbei handelt es sich lediglich um eine optische Option, die nach Wunsch frei modifiziert werden kann.

▷ noshowallfields (showallfields)

Wird showallfields gesetzt, so werden auch dann Felder in env@requirement → p. 3 angezeigt, wenn sie keinen Wert erhalten haben.

2.2 Farben

Mit usetabu definiert die Farbe HeaderColor die Farbe der Titelspalte und NextHeaderColor die Farbe im Falle eines Seitenumbruchs. Die Farben können wie jede andere geändert werden. Innerhalb von sopra–base wird automatisch eine dem Design angepasste Farbe gewählt.

3 Befehle- und Umgebungen

Es gilt zu beachten, dass das Präfix env@ nur auf die Natur einer Umgebung hinweist und nicht zum eigentlichen Bezeichner zuzuordnen ist!

Weiter gilt: Damit alle Titel und Längen richtig aufgelöst werden können, muss das Dokument in der Regel zwei mal kompiliert werden um eine korrekte Anzeige zu erzeugen.

3.1 Definition von Anforderungen

▷ env@requirements*[Start–ID]*{functional/quality}

Startet einen neuen Block an Anforderungen, wobei innerhalb von hier env@requirement → p. 3 zur Verfügung steht. Standardmäßig werden die Anforderungen von 1 an für das ganze Dokument durchnummeriert, sollte dieser Startwert allerdings geändert werden (wofür es, abseits von Testzwecken eigentlich keinen Sinn geben sollte, da es so manche Anforderungen doppelt geben kann!) so ist dies durch das Setzen von Start–ID möglich. Verpflichtend muss angegeben werden, ob es sich um *funktionale* (functional) oder *non-funktionale* (quality) Anforderungen handelt. Alle hierrin verschatelten Anforderungen werden als solche gewertet. Die beiden Varianten besitzen ihre

eigenen Zähler!. Entwicklernotiz: Die benötigten Werte sind in \@sor@fnctl und \@sor@qltl gespeichert. Das Präfix entsprechend in \sor@functional@requirementprefix und \sor@quality@requirementprefix.

▷ env@requirement*[ID]*{Bezeichner}

Diese Umgebung steht nur innerhalb von env@requirements→ p. 2 zur Verfügung! Hiermit kann eine einzelne Anforderung definiert werden, wobei die ID sich entweder sinnvoll durch ein hochzählen des Zählers oder durch die explizite Angabe ergibt. Eine ID muss stets numerisch sein. Weiter kann hier ein Bezeichner angegeben werden, wobei dieser Standardmäßig den Titel der Anforderung repräsentiert, solange dieser nicht explizit mittels \setTitle→ p. 3 geänert wird. Ein Bezeichner darf keine Umlaute oder andere Sonderzeichen enthalten. Auch Formatierungs- und andere LaTeX 2ε/TeX-Befehle können hier nicht verwendet werden. Der {Bezeichner} kann ebenfalls dafür verwendet werden um mittels \preqref→ p. 4 auf diese Anforderung zu referenzieren. Es darf keinen doppelten Bezeichner geben.

3.2 Daten in einer Anforderung setzen

Die folgenden Befehle können nur innerhalb von env@requirement→ p. 3 verwendet werden! Es steht noch aus an einem Verfahren zu arbeiten, welches es erlaubt eigene Felder zu definieren. Es ist geplant translate zu verwenden um zumindest die Datenfelder übersetzen beziehungsweise allgemein frei definieren zu können!

▷ \setDescription{Beischreibungstext}

Setzt eine Erklärung zu der jeweiligen Anforderung. Hier kann prinzipiell alles gesetzt werden, von einer env@eagle-map über env@itemize bis hin zu ganz normalem LaTeX.

▷ \setJustification{Begründungstext}

Setzt die Begründung für die jeweilige Anforderung. Es gilt bezüglich des Inhalts das gleiche wie für \setDescription→ p. 3.

▷ \setDependencies{Kommaseparierte Liste}

Erlaubt das auflisten aller Abhängigkeiten mit von \preqref→ p. 4 akzeptierten Schemata. So zum Beispiel: \setDependencies{FA1,QA42,main-menu}.

▷ \setTitle{Titel}

Erlaubt es den Titel der Anforderung zu ändern. Dies ändert nicht den in env@requirement→ p. 3 gesetzten Bezeichner! (Dies kann durch showtag) verdeutlicht werden.

▷ \setPriority{Zahl}

Erlaubt es die Priorität der jeweiligen Anforderung zu definieren. Die Zahl muss zwischen (inklusiv) 0 und 5 liegen.

3.3 Allgemeine Befehle

▷ \thesorversion

Liefert die aktuelle Version des Pakets. So ergibt: \thesorversion: v1.0.09
Hinweis: über \value{sorversion} lässt sich die Version als 4-stellige Nummer erhalten: 1009.

▷ `\iskip`

Setzt einen *Negativabstand*, der dann verwendet werden kann, wenn einem ein Abstand zu groß gesetzt ist. Dies wird nur dann nötig sein, wenn man eigene Umgebungen oder vergleichbares einbringt und ist hier nur eine Art Inspiration.

▷ `\preqref[Text]{Anforderungsmarker}`

Dieser Befehl setzt einen Verweis auf eine mittels `env@requirement` → p. 3 gesetzt Anforderung. Für jede werden zwei gültige Marker generiert:

- Es kann der Bezeichner verwendet werden.
- Es kann das Kürzel für den Anforderungstyp plus die jeweilige Nummer verwendet werden (`FA13`, `QA9`, ...)

Dieser Befehl kann auch mit einem Sternchen benutzt werden. In diesem Fall wird der Bezeichner der Anforderung nicht ausgegen, sondern nur das Kürzel (zum Beispiel `FA42`) samt der Seite auf der die Anforderung definiert wird. Wird `Text` gesetzt, so wird dieser anstelle des Anforderungsnamen angezeigt.

▷ `\pref{Hypermarker}{Text}`

Wird von `\preqref` → p. 4 verwendet und gibt einfach nur einen Link mit Seitenzahl an. Kann modifiziert(/überschrieben) werden um das Verhalten von `\preqref` → p. 4 anzupassen.

▷ `\sorSetPriority{Zahl}`

Funktioniert analog zu `\setPriority` → p. 3, kann allerdings auch außerhalb von `env@requirement` → p. 3 verwendet werden.

3.4 Beispiele

Definieren wir uns im Folgenden einmal zwei Anforderungen:

```
\begin{requirements}{functional}
   \begin{requirement}{a-duck}
      \setTitle{Ente}
      \setDescription{Es muss Enten geben. Damit sie am Leben bleiben, gibt es \preqref{feed-ducks}}
      \setJustification{Weil sie Knuffig sind}
   \end{requirement}

   \begin{requirement}{feed-ducks}
      \setTitle{Ente füttern}
      \setDescription{Es muss möglich sein, die Enten aus \preqref{FA1} zu füttern. Wir haben: \preqref[Enten]{FA1}.}
      \setJustification{Enten müssen Leben (dürfen)!!}
      \setDependencies{a-duck}
      \setPriority{4}
   \end{requirement}
\end{requirements}
```

Das Ergebnis ist (mit `tabu`) das folgende:

ID	FA1	a-duck
TITEL:	Ente	
BESCHREIBUNG:	Es muss Enten geben. Damit sie am Leben bleiben, gibt es FA2: 'Ente füttern' → p. 5	
BEGRÜNDUNG:	Weil sie Knuffig sind	

ID	FA2	feed-ducks
TITEL:	Ente füttern	★★★★☆
BESCHREIBUNG:	Es muss möglich sein, die Enten aus FA1: 'Ente' → p. 4 zu füttern. Wir haben: Enten (FA1) → p. 4.	
BEGRÜNDUNG:	Enten müssen Leben (dürfen)!!	
ABHÄNGIGKEITEN:	a-duck → p. 4	