Die TEXnische Komödie

dante
Deutschsprachige
Anwendervereinigung TEX e.V.

Hendri Adriaens, Uwe Kern: *Neue Mechanismen in der Verarbeitung von Keys*, Die TEXnische Komödie 3/2005, S. 47–58.

Neue Mechanismen in der Verarbeitung von Keys

Hendri Adriaens, Uwe Kern

Dieser Artikel[1] soll in das (LA)TEX-Paket xkeyval einführen, welches eine Erweiterung des bekannten Pakets keyval ist. Das neue Paket bietet flexiblere Kommandos, Syntaxerweiterungen und einen neuen Mechanismus zur Verarbeitung von Klassen- und Paket-Optionen, der sich die Syntax *key=value* zu Nutze macht.

Einführung

Das Paket keyval [2] von David Carlisle wird häufig von Paket-Autoren verwendet, um eine große Zahl optionaler Argumente in Makros einzubauen. Die Vorteile von keyval sind, dass zum einen die Zahl optionaler Argumente nicht mehr auf neun limitiert ist und dass zum anderen die Argumente Namen erhalten und so die Syntax eines Makros vereinfachen.

[1] Erstveröffentlichung erfolgte in *MAPS* 31 (2004). Dieser um Zusätze erweiterte Nachdruck erfolgt mit freundlicher Genehmigung der Autoren. Die Übersetzung erledigte Johannes Reinhardt.

Mit dem Paket können so genannte „Key-Makros“ definiert werden, die die Benutzereingaben verarbeiten. Durch `\KV@family@keyname` werden diese „Key-Makros“ definiert, wobei `KV` ein festes Präfix ist, das Kollisionen mit anderen Paketen vermeidet. Diese Makros haben ein Argument, um Benutzereingaben zu verarbeiten. Ein Makro für den Key `pi` kann z. B. definiert werden über

```
\define@key{myfam}{pi}{\setlength{\parindent}{#1}}
```

Diese Zeile definiert ein Makro namens `KV@myfam@pi`. Solche Key-Makros werden aufgerufen, wenn `\setkeys` verwendet wird, um die Keys zu setzen. In unserem Fall wird das Key-Makro `\parindent` auf den entsprechenden Wert gesetzt, wenn `pi` benutzt wird. Hier ein typisches Beispiel für die Anwendung:

```
\setkeys{myfam}{pi=10pt,pn=Page~\thepage}
```

Die Pakete keyval und xkeyval richten sich in erster Linie an Klassen- und Paketautoren. Die `\define@key`-Kommandos werden gewöhnlich in der Dokumentenpräambel oder einem Paket verwendet und der Anwender benutzt `\setkeys`. Die Datei `xkeyval.tex` kann mit Plain TEX verwendet werden. Alle Features, die hier beschrieben werden, sind auch dort verfügbar, mit Ausnahme des „`X`“-Makros auf S. 49.

Warum ein neues Paket?

Beim Arbeiten an einem anderen Paket entstand der Bedarf, mehrere Familien von Keys zur Verfügung zu haben. Jede Familie sollte Keys für ein spezielles Makro oder eine spezielle Umgebung liefern. So wäre man in der Lage, die Verwendung unzulässiger Keys, die zerstörerische Auswirkungen auf den Rest des Dokuments haben könnten, innerhalb der Makro-Argumente zu verhindern. Nett wäre es auch, dem Benutzer erlauben zu können, gewisse Keys eines Makros oder einer Umgebung global in der Präambel festzulegen. So könnte man beispielsweise die Vorgaben zum Aussehen aller `example`- und `exercise`-Umgebungen des Dokumentes in der Präambel erlauben, aber gleichzeitig verbieten, das Aussehen lokal zu verändern, und umgekehrt. In etwas komplizierteren Fällen kann die Verwendung von Keys in Makros, die für den Umgang mit diesen Keys nicht gedacht sind, zu Fehlern führen, die kaum noch nachzuvollziehen sind. Das war der Anfang des Pakets xkeyval [1].

Als wir dabei waren, keyval zu verallgemeinern, bemerkten wir, dass es bereits eine Menge Pakete gab, die gewisse Fähigkeiten erweitern, alle auf ihre eigene

Art. Einige liefern z. B. ein System für den Gebrauch von Keys und Werten in `\usepackage`-Kommandos. Die bekanntesten Beispiele sind die Pakete hyperref, geometry und beamer. All diese Ansätze unterscheiden sich in Details und lassen sich nicht ohne weiteres Programmieren auf andere Pakete übertragen. Das verlangt nach einer einheitlichen Lösung.

Eine besondere Fähigkeit, die beispielsweise in hyperref verwendet wurde, ist die Verfügbarkeit von Keys des Typs `boolean`, die nur die Werte `true` oder `false` annehmen können. hyperref implementiert das über `\define@key` im gewöhnlichen Key-System. Weil aber die Funktion (oder zumindest ein Teil), die bei Verwendung des Keys aufgerufen werden soll, schon vorher bekannt ist (nämlich das einfache Setzen eines „if“-Kommandos je nach Aufruf auf `true` oder `false`), kann das System vereinfacht werden.

Ein letzter Grund, das neue Paket zu entwickeln, war die Tatsache, dass die Entwicklung des Pakets keyval offenbar seit 1999 stillstand und fundamentale Änderungen und Verbesserungen am System mit einem neuen Paket einfacher zu realisieren waren. Die Verbesserungen betreffen unter anderem Makros, die Paket-Optionen erzeugen, mit denen Werte übergeben werden können. Es gibt neue Arten von Keys, eine Zeiger-Syntax und ein System für vordefinierte Werte. Mehrere Familien in `\setkeys` und robustes Eingabe-Parsing werden akzeptiert. Schließlich werden die Pakete aus der PSTricks-Serie unterstützt. In den folgenden Abschnitten dieses Artikels werden diese Entwicklungen behandelt.

Keys und Werte in Paket-Optionen

Zu allererst stellt das Paket Makros zur Verfügung, um Klassen- oder Paket-Optionen zu deklarieren, auszuführen und zu verarbeiten. Die Makros sind unter den üblichen LaTeX-Namen verfügbar, haben aber alle das Suffix `X`, nämlich

```
\DeclareOptionX
\DeclareOptionX*
\ExecuteOptionsX
\ProcessOptionsX
```

Mit diesen Kommandos kann der Benutzer einer Option einen Wert zuordnen, genauso wie mit `\setkeys`. Das erste Makro basiert auf `\define@key`, die letzten beiden basieren auf `\setkeys`. Wenn das Paket mypack die Befehle kennt, könnte ein Benutzer beispielsweise Folgendes machen:

```
\usepackage[textcolor=red,font=times]{mypack}
```

Diese Makros sind vollständig in das System der LaTeX-Optionen integriert. So können Pakete globale Optionen aus dem `\documentclass`-Befehl kopieren, Optionen an andere Klassen oder Pakete weiterreichen und die Liste der unbenutzten, globalen Optionen aktualisieren, die dann von LaTeX im Logfile ausgegeben wird.

Wertzuweisungen wie `author=\textit{Me}` in Klassen- und Paket-Optionen sind jedoch nicht erlaubt, obwohl sie leicht von `\setkeys` verarbeitet werden könnten. Diese Einschränkung resultiert aus der Art, wie LaTeX Optionen verarbeitet. Dabei wird die gesamte Liste von Optionen (Keys und Werte) komplett expandiert, was offensichtlich Probleme verursacht, denn `author=\protect\textit{Me}` wäre *keine* Lösung für dieses Problem.

Um diese verfrühte Expansion zu vermeiden, müssen diverse Kernel-Makros umdefiniert werden. xkeyval beinhaltet das Paket xkvltxp, welches die neuen Definitionen enthält. Lädt man das Paket bevor man die Klassen oder Pakete einbindet, die xkeyval nutzen, so können die Klassen- und Paket-Optionen expandierbare Makros enthalten. xkvltxp wird nicht in den LaTeX 2_ε-Kernel eingebaut, weil es Kompatibilitätsschwierigkeiten bei Anwendern verursachen könnte, die einen alten Kernel zusammen mit neuen Paketen benutzen, die diese veränderte Funktionalität voraussetzen.

Präfixe, Familien, Keys und Zeiger

Das Paket xkeyval liefert eine erweiterte Syntax für alle Befehle aus keyval. Die Dokumentation zu xkeyval erläutert syntaktische Details, die in diesem Artikel nicht diskutiert werden. Der Befehl, mit dem Keys definiert werden, wurde um ein optionales Argument erweitert, das dem Key-Makro als Präfix vorangestellt wird. Alle Paket-Autoren sollten ein paketspezifisches Präfix für alle internen Makros verwenden, um nicht versehentlich Makros anderer Pakete umzudefinieren. Mit diesem Argument können auch in spezialisierten Systemen wie PSTricks Keys definiert und gesetzt werden. Darauf werden wir im Abschnitt über das Paket pst-xkey zurückkommen.

Die Syntax für das Setzen von Keys mit `\setkeys` wurde entsprechend angepasst. Wie in der Einführung erwähnt, kann man auch eine Liste von Familien angeben, die beim Setzen von Keys durchsucht werden sollen:

```
\setkeys{font,page}{fs=10pt,pn=Page~\thepage}
```

Das Paket bietet auch neue Typen von Keys. Dabei handelt es sich um Auswahl-Keys, die eine beschränkte Zahl möglicher Eingabewerte zulassen und um boolesche Keys, die ausschließlich die Werte `true` und `false` annehmen können, wie im folgenden Beispiel:

```
\define@choicekey{fam}{keya}{\fbox,\mbox}{#1{text}}
\define@boolkey{fam}{keyb}{%
  \ifKV@fam@keya we continue\else we stop here\fi}
\setkeys{fam}{keya=\mbox,keyb=false}
```

Diese Keys erzeugen eine Fehlermeldung, wenn der Benutzer einen Wert bestimmt, der nicht erlaubt ist. Weitere Erläuterungen dazu enthält die Dokumentation. Das Paket bietet mit xkvview ein Werkzeug, um Tabellen mit Informationen über die definierten Keys zu erstellen.

Mit der neuen Syntax können auch Zeiger auf Keys verweisen. Mit Zeigern kann man z. B. `keyb` den Wert zuweisen, der `keya` zugewiesen wurde, völlig egal, was dieser Wert ist. Beispielsweise

```
\setkeys{family}{\savevalue{keya}=red,keyb=\usevalue{keya}}
```

`\savevalue` veranlasst an dieser Stelle xkeyval dazu, sich den Wert zu merken, der `keya` zugewiesen wurde. `\usevalue` verwendet diesen Wert. (Man kann sich mit dem `\savekeys`-Befehl das wiederholte Tippen von `\savevalue` sparen.) Wenn in diesem Beispiel `red` auf `blue` umgestellt wird, so müssen keine Änderungen mehr an `keyb` vorgenommen werden, damit diesem ebenfalls `blue` zugewiesen wird. Ähnlich verarbeitet TEX die Definition `\def\cmdb{\cmda}`.

Diese Zeiger können auch für das Setzen von Standardwerten verwendet werden. Sie weisen dem Key-Makro einen Standardwert zu, sobald der Benutzer einen bestimmten Key gebraucht, ihm aber keinen Wert zuweist. So könnte man z. B. definieren:

```
\define@key{fam}{keya}{keya: #1 }
\define@key{fam}{keyb}[\usevalue{keya}]{keyb: #1 }
```

Dann würde der folgende Aufruf von `\setkeys`

```
\setkeys{fam}{\savevalue{keya}=test,keyb}
```

das unten stehende Ergebnis liefern:

```
keya: test keyb: test
```

Wir werden nun ein paar technische Details zur Syntax der Zeiger diskutieren. Zu Beginn sind die Kontrollsequenzen `\savevalue` und `\usevalue` noch nicht definiert! Statt dessen nutzt das Paket sie als Platzhalter. Ein einfacher Parsing-Schritt entscheidet, ob `\savevalue` im Namensteil des Keys benutzt wurde. Parsing wird auch genutzt, um das Auftreten von `\usevalue` durch den gespeicherten Wert zu ersetzen. Wird ein Zeiger ersetzt, so wird dieser Ersatz wiederum auf Zeiger durchsucht, die gegebenenfalls ersetzt werden. Das ermöglicht verschachtelte Zeiger in Key-Werten. Außerdem stellt dieses Parsing sicher, dass ein Wert keine Zeiger mehr enthält, wenn er dem Key-Makro übermittelt wird – solange der Zeiger nicht in einer Gruppe vor xkeyval versteckt wurde.

Wenn der Benutzer dem Key keinen Wert zugewiesen hat, ist der Ersetzungsprozess ein bisschen vertrackter. In diesem Fall soll der Standardwert des Keys (wenn vorhanden) nach Zeigern durchsucht werden. Standardwert-Makros sehen so aus:

```
\def\prefix@fam@key@default{\prefix@fam@key{the default value}}
```

Das Makro `\prefix@fam@key@default` wird ausgeführt, wenn der Benutzer dem Key keinen Wert zuweist. Dieses System wurde von keyval eingeführt und viele Pakete machen davon Gebrauch. Einige Pakete nutzen es aber nicht auf die Weise, wie es in keyval gedacht ist. Zum Beispiel definiert das Paket fancyvrb Standardwert-Makros, die beliebigen Code an Stelle des vorgesehenen `\prefix@fam@key` ausführen. Um die Kompatibilität zu gewährleisten, muss dieses Verhalten unterstützt werden, obwohl dadurch eine saubere Lösung des Problems verbaut ist. Beispielsweise könnten wir sonst `\prefix@fam@key@default` von Beginn an als „den Standardwert“ definieren – ohne getrennten Makroaufruf.

Das ist eine wichtige Einschränkung des Zeiger-Systems, da wir den Standardwert vom Standardwert-Makro erhalten und es nach Zeigern absuchen wollen. Deshalb verfährt xkeyval folgendermaßen: Es prüft zuerst, ob das Standardwert-Makro wie erwartet startet, nämlich mit einem Key-Makro `\prefix@fam@key`. Ist das der Fall, definiert es lokal das Key-Makro um und sichert den Wert in einem temporären Makro, um es anschließend auszuführen. Das temporäre Makro enthält dann den Standardwert, der nach Zeigern durchsucht werden kann. Ist das Standardwert-Makro nicht von der erwarteten Art, wie bei fancyvrb, dann führt xkeyval es aus, ohne zu versuchen, den Standardwert zu erhalten oder Zeiger zu ersetzen.

Voreingestellte Werte

Das System der Standardwerte kommt zum Einsatz, wenn Benutzer Keys angeben, diesen Keys aber keine Werte mitgeben. Das Paket keyval bietet ursprünglich keinen Weg, um auch solchen Keys, die überhaupt nicht vom Benutzer verwendet werden, einen Wert zuzuweisen. In vielen Anwendungsfällen will man Standardwerte für Keys einsetzen, wenn diese nicht explizit benutzt werden. Beispielsweise „Skaliere diese Abbildung mit Faktor 1 wenn der Benutzer nichts anderes sagt“. Nun könnte man einfach loslegen, das Key-Makro mit einer Voreinstellung aufrufen und nachher die Benutzereingabe an `\setkeys` weiterleiten, um schließlich die Werte, die man gerade gesetzt hat, wieder zu überschreiben. In Fällen, wo Keys nicht selbstständig Informationen generieren, sondern z. B. nur eine Länge setzen, ist das möglich (aber recht schwerfällig, wenn man viele Keys hat).

Was passiert aber, wenn wir dieses Schema auf Keys anwenden, die folgendermaßen definiert sind?

```
\define@key{fam}{keya}{Your input was: #1}
\define@key{fam}{keyb}{\edef\list{\list,#1}}
```

Wenn wir dem selben Schema folgen wie im ersten Beispiel, so würde sowohl unsere Voreinstellung als auch unsere Benutzereingabe (wenn vorhanden) in der Ausgabe erscheinen. Im zweiten Beispiel werden die Voreinstellung und die Benutzereingabe an die Liste in `\list` angehängt.

Um das zu vermeiden, verwaltet xkeyval Voreinstellungen. Zuerst deklariert man die Keys, die voreingestellt werden sollen, und ihre Werte über `\presetkeys`, z. B.:

```
\savekeys{fam}{head}
\presetkeys{fam}{head=red}{tail=\usevalue{head}}
```

Der Grund, weshalb das Makro `\presetkeys` zwei Argumente hat, die Voreinstellungen für Keys enthalten, wird gleich verständlich werden.

Wenn jetzt Benutzereingaben die Keys in der Familie `fam` festlegen, stellt das Makro `\setkeys` fest, welche Keys vom Benutzer gesetzt werden und füllt diese *nicht* mit Voreinstellungen. Keys, die der Benutzer nicht festlegt, werden dann auf die Werte gesetzt, die in `\presetkeys` eingestellt sind.

Wenn man Zeiger einsetzt, sollte man eine Sache unbedingt im Hinterkopf behalten: Zeigt der Zeiger auf einen Key, der erst später einen Wert zugewie-

sen bekommt, so kann der Zeiger den Wert zu diesem Zeitpunkt nicht wissen und es kommt zu Fehlern. Daher ist es meistens, wie in dem Beispiel oben, am besten, Voreinstellungen mit Zeigern erst ganz am Ende auszuführen.

Eine ähnliche Unstimmigkeit kann auftreten, wenn Keys ohne Zeiger in ihrem Wert *nach* dem Setzen der Benutzereingaben eine Voreinstellung zugewiesen bekommen. Benutzer können dann im späteren Verlauf der Ausführung keine Zeiger mehr auf diese Voreinstellungen verwenden. Deshalb sollte man Keys ohne Zeiger mit ihrem Wert ganz am Anfang ausführen, noch bevor Benutzereingaben gesetzt werden.

Das ist auch der Grund, warum das Makro `\presetkeys` zwei Argumente hat: Das erste (das gewöhnlich Keys und Werte ohne Zeiger enthält) wird eingefügt, bevor vom Benutzer eingegebene Keys gesetzt werden, das zweite (das Zeiger zu Voreinstellungen oder Benutzereingaben enthält) danach.

Dieses Vorgehen ist insbesondere dann nützlich, wenn man sich nicht darauf verlassen kann, dass Werte von Keys lokal in einem Makro oder einer Umgebung bleiben, weil das System für Voreinstellungen die Werte der Keys bei jedem Aufruf des Makros oder der Umgebung auf die Voreinstellung zurücksetzt, wenn der Benutzer diese nicht lokal überschrieben hat. Wir werden das näher erläutern: `\def`-Definitionen (beispielsweise erstellt von Key-Makros) werden von TEX beim Verlassen einer Gruppe oder Umgebung zerstört. Deshalb bleiben solche Werte lokal. Wenn allerdings die Keys nicht immer `\def` benutzen, sondern z. B. `\gdef`, dann durchbrechen diese globalen Definitionen ihre Gruppe oder Umgebung und können alle folgenden Makros und Umgebungen durcheinander bringen. Daher muss man die Werte der Keys bei jedem Gebrauch des Makros oder der Umgebung sorgfältig neu initialisieren.

All das ist mit dem System der Voreinstellungen nicht mehr notwendig. Wenn erst einmal die Voreinstellungen für eine bestimmte Familie definiert wurden, werden jedes Mal, wenn die Familie in `\setkeys` verwendet wird, die Voreinstellungen zusammen mit den Benutzereingaben berücksichtigt.

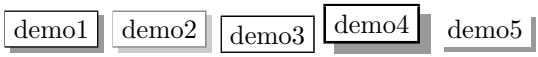
Das folgende Beispiel zeigt die Stärke einer Kombination von Voreinstellungen und Zeigern. Unter dem Beispiel folgt die Ausgabe und eine Erklärung. Nehmen wir an, dass wir ein einfaches Kommando für eine Box mit Rahmen und Schatten mit folgendem *Standard*verhalten erstellen wollen:

- Ein Schatten wird nur dann gezeichnet, wenn die Box einen Rahmen hat.

- Die Schattenfarbe soll 40 % der Farbe des Rahmens haben, um klar unterscheidbar zu sein.
- Der Schatten soll viermal so breit wie der Rahmen sein.

Selbstverständlich soll der Benutzer jede dieser Regeln überschreiben können, wenn er das Box-Kommando verwendet.

```
1  \documentclass{article}
2  \usepackage{xkeyval}
3  \usepackage{calc,xcolor}
4  \makeatletter
5  \newdimen\shadowsize
6  \define@boolkey{Fbox}{frame}[true]{}
7  \define@boolkey{Fbox}{shadow}[true]{}
8  \define@key{Fbox}{framecolor}{\def\Fboxframecolor{#1}}
9  \define@key{Fbox}{shadowcolor}{\def\Fboxshadowcolor{#1}}
10 \define@key{Fbox}{framesize}{\setlength\fboxrule{#1}}
11 \define@key{Fbox}{shadowsize}{\setlength\shadowsize{#1}}
12 \savekeys{Fbox}{frame,framecolor,framesize}
13 \presetkeys{Fbox}{frame,framecolor=black,framesize=0.5pt}%
14  {shadow=\usevalue{frame},shadowcolor=\usevalue{framecolor}!40,
15   shadowsize=\usevalue{framesize}*4}
16 \newcommand*\Fbox[2][]{%
17  \setkeys{Fbox}{#1}{\ifKV@Fbox@frame\else\fboxrule0pt\fi
18   \ifKV@Fbox@shadow\else\shadowsize0pt\fi
19   \sbox0{\fcolorbox{\Fboxframecolor}{white}{#2}}%
20   \hskip\shadowsize
21   \color{\Fboxshadowcolor}\rule[-\dp0]{\wd0}{\ht0+\dp0}%
22   \llap{\raisebox{\shadowsize}{\box0\hskip\shadowsize}}}}
23 \makeatother
24 \begin{document}
25 \Fbox{demo1} \Fbox[framecolor=gray]{demo2}
26 \Fbox[shadow=false]{demo3} \Fbox[framesize=1pt]{demo4}
27 \Fbox[frame=false,shadow]{demo5}
28 \end{document}
```



Zuerst werden in den Zeilen 6 bis 11 die Keys definiert, die wir in diesem Beispiel brauchen. Das Kommando `\presetkeys` in Zeile 13 definiert die Voreinstellungen: Der Rahmen wird auf `true` gesetzt, seine Farbe auf Schwarz

und seine Rahmengröße auf 0.5 pt, es sei denn, der Benutzer liefert andere Einstellungen zu diesen Keys. Die oben aufgelisteten Anforderungen werden durch die Zeiger-Ausdrücke im nächsten Argument abgedeckt.

Die erste Anwendung der Box zeigt die Standard-Box ohne zusätzliche Benutzereingaben. Wir sehen einen Rahmen und einen Schatten, alles basierend auf der Farbe schwarz. Die zweite Box zeigt, wie die Benutzereingaben für den Rahmen die voreingestellten Werte überschreiben. Die Box wird grau. Weil aber die Schattenfarbe standardmäßig von der Rahmenfarbe abgeleitet wird, wird der Schatten hellgrau. Im dritten Beispiel haben wir einen Rahmen, aber keinen Schatten. Zu beachten ist, dass die Rahmenfarbe wieder schwarz – die Voreinstellung – ist. Die vierte Box hat eine erhöhte Rahmenbreite und durch den Gebrauch des Zeigers beim Voreinstellen der Keys auch eine größere Schattenbreite. Das letzte Beispiel zeigt, dass es möglich ist, das voreingestellte Verhalten zur Verbindung zwischen Rahmen und Schatten vollkommen zu überschreiben: Es gibt einen Schatten ohne Rahmen aus.

Robustes Parsen

Genau wie die Zeiger-Platzhalter `\savevalue` und `\usevalue`, behandeln keyval und xkeyval auch das Komma und das Gleichheitszeichen als Platzhalter. In der Vergangenheit hat das zu Problemen geführt. So gibt es eine bekannte Inkompatibilität zwischen den Einstellungen des Pakets babel für die türkische Sprache und allen Paketen, die keyval benutzen. Weil das türkische babel den Catcode des Gleichheitszeichens zu Gunsten einer Abkürzung verändert, können die Parser-Makros aus keyval diese Zeichen nicht mehr erkennen und produzieren Fehlermeldungen. Weitere Information zu diesem Problem mit keyval und babel gibt es bei http://www.latex-project.org/cgi-bin/ltxbugs2html?pr=babel/3523.

xkeyval löst dieses Problem, indem es alle beim Parsen benötigten Zeichen säubert (d. h. den catcode auf 12 setzt). Diese Aufgabe übernimmt das Makro `\@selective@sanitize`, das ein einzelnes oder mehrere Zeichen in einem einzigen Aufruf säubern kann. Auch die Tiefe der Säuberung kann angegeben werden. xkeyval implementiert das Makro so, dass nur Kommas und Gleichheitszeichen in der obersten Ebene eines Key-Werts gesäubert werden. Das ist schließlich alles, was man braucht, um die Eingabe zu parsen. Zeichen innerhalb von Gruppen bleiben unberührt, und man kann deshalb sogar babel-Kurznotationen verwenden ohne Fehler zu produzieren:

```
\usepackage[turkish]{babel}
```

```
... \setkeys{fam}{key={some =text}}
```

In diesem Beispiel wird das erste „=“ zum Parsen gesäubert, während das zweite „=“ unberührt bleibt und damit seine ursprüngliche Bedeutung behält.

Makros umdefinieren?

Bestehende Makros umzudefinieren ist im Allgemeinen gefährlich. Trotzdem definiert das Paket xkeyval die zwei keyval-Hauptmakros `\define@key` und `\setkeys` um. So verhindern wir, dass verschiedene, parallel laufende Systeme ähnliche Dinge tun und Verwirrung stiften.

Obwohl xkeyval die gesamte Syntax des Originalpakets keyval unterstützt, mussten wir die Pakete prüfen, die keyval benutzen, bevor wir uns entscheiden konnten, die Makros umzudefinieren. Drei Hauptfragen kamen in diesem Prozess auf.

Zuerst mussten wir feststellen, dass ein paar Pakete interne Teile von keyval statt der Benutzerschnittstelle über `\define@key` und `\setkeys` verwenden. Um jegliche Fehler wegen nicht definierter Kontrollsequenzen zu vermeiden, lädt xkeyval also die Interna von keyval, wenn keyval nicht schon vorher geladen wurde. Zweitens benutzten bestimmte Pakete das Standardwert-System auf kreative Weise. Im Abschnitt über die Zeiger-Syntax haben wir das Problem dargestellt und xkeyvals Lösung erläutert.

Schließlich haben wir entdeckt, dass pst-key die Makros `\define@key` und `\setkeys` umdefinierte, um das Setzen von Keys in PSTricks zu ermöglichen. Wir haben diesen Punkt mit Herbert Voß, dem Maintainer von PSTricks diskutiert, und sind zu dem Schluss gekommen, dass xkeyval einen einheitlichen Zugriff auf Keys und Werte entwickeln sollte, um so pst-key abzulösen. Mehr Informationen über die Entwicklung von PSTricks gibt es im letzten Abschnitt.

Nachdem die nötigen Makros umdefiniert sind, stellt xkeyval sicher, dass das Paket keyval nicht nachträglich geladen werden kann. So wird verhindert, dass die xkeyval Makros noch einmal umdefiniert werden. Dies ist der letzte Schritt, um die keyval-Makros sicher umzudefinieren und ein System anzubieten, auf das alle Paket-Autoren ihre Pakete ohne allzu großen Aufwand umstellen können.

Das Paket pst-xkey

Eine entscheidende Reihe von Paketen wird in naher Zukunft xkeyval benutzen, nämlich die PSTricks-Pakete [3, 4]. Für das Verarbeiten von Keys und Werten verwenden sie momentan noch eine Kombination aus eigenen Definitionen in `pstricks.tex` und pst-key, wobei letzteres eine Modifikation des Pakets keyval darstellt.

Wegen der Beliebtheit und Flexibilität des Pakets PSTricks haben mehrere Nutzer Erweiterungen zur Originaldistribution beigetragen. Unglücklicherweise hatten alle Keys in PSTricks dieselbe Form, nämlich `\psset@somekey`. Deshalb mussten PSTricks-Autoren alle bestehenden Pakete durchsehen, um nicht irgendeinen bestehenden Key umzudefinieren.

Herbert Voß hatte das Problem erkannt und bald begann die Arbeit, xkeyval den Weg zu ebnen, um damit in PSTricks-Keys zu definieren und zu setzen. Wenn Paket-Autoren ihre Keys in eine ausgewählte Familie (z. B. mit dem Paketnamen) verpacken, dann brauchen sie keine anderen Pakete nach bestehenden Keys abzusuchen. Darin liegt der entscheidende Vorteil von xkeyval.

Um das zu ermöglichen, mussten `\define@key` und `\setkeys` so angepasst werden, dass der Standardpräfix `KV` von keyval verändert werden kann, z. B. in `psset`. Dann musste das Makro `psset` umdefiniert werden, um das neue `\setkeys` zu benutzen und alle verfügbaren Familien zu suchen. Wenn ein Paket aus PSTricks geladen wird, fügt es alle Familien, die in diesem Paket benutzt werden, einer Liste hinzu. Diese Liste wird dann in `\setkeys` verwendet. Weil alle unterschiedlichen Pakete verschiedene Familien benutzen, ist die Wiederverwendung von Namen für Keys bald kein Problem mehr. Die neue Definition von `\psset`, ebenso wie ein paar andere Makros, die für diese Aufgabe benötigt werden, ist im Paket `pst-xkey` enthalten, das zusammen mit xkeyval verteilt wird.

Literatur

[1] Hendri Adriaens: *xkeyval package, v2.4, 2005/03/31*; `CTAN:/macros/latex/contrib/xkeyval`.

[2] David Carlisle: *keyval package, v1.13, 1999/03/16*; `CTAN:/macros/latex/required/graphics`.

[3] Herbert Voß: *PSTricks web site*; `http://www.pstricks.de`.

[4] Timothy Van Zandt et al.: *PSTricks package, v1.04, 2004/06/22*; CTAN:/graphics/pstricks.