Partie II
KOMA-Script pour les utilisateurs avancés et les experts

Dans cette partie, sont regroupées des informations sur les classes LaTeX, à destination des créateurs de packs. Elles précisent non seulement les instructions utiles pour la mise en œuvre des packs et classes, mais aussi les interfaces nécessaires à la création avec KOMA-Script. Elle comporte, en outre, des précisions sur les options obsolètes ainsi que des notes de fond sur l'optimisation de KOMA-Script.
Le chapitre II est destiné à compléter les informations à destination des auteurs d'articles, de rapports, de livres et de lettres, avec plus de notes et d'avantage d'exemples pour les utilisateurs.

12. Les fonctions de base du pack scrbase

Le pack scrbase fournit des fonctionnalités de base conçues et mises en œuvre à l'usage des auteurs de packs et de classes. Il peut aussi être utilisé pour les classes wrapper liées ou non à la classe KOMA-Script. Ainsi les auteurs de classes étrangères à KOMA-Script bénéficient de toutes les fonctionnalités de scrbase.

12.1. Charger le pack

Un utilisateur charge un pack en employant \usepackage, un auteur utilisera \Requirepack. Les packs d'encapsulation tels que wrapper nécessitent \RequirepackWithOptions. La commande \Requirepack a les mêmes arguments optionnels que \usepackage.
En revanche, \RequirepackWithOptions n'a pas un argument facultatif, mais passe toutes les options proposées lors du chargement du pack de wrapper pour le pack requis. Voir [Tea06] pour plus d'informations sur ces commandes.

Le pack scrbase nécessite, en interne, l'utilisation des fonctionnalités du pack keyval qui peuvent être fournies alternativement par le pack xkeyval. Le pack keyval gère les définitions de clés et les attributions de valeurs à ces clés. Il est possible d'utiliser, avec scrbase, la syntaxe keyval: key=value (clé=valeur).

internalonly=valeur

Le pack Scrbase fournit pour une exécution conditionnelle, des commandes internes dont les noms primaires ressemblent à \scr@nom, utilisées également et uniquement en interne par KOMA-Script. L'auteur de packs et de classes peut utiliser ces commandes mais ne devrait pas les redéfinir car d'autres packs peuvent comporter des ordres avec le même nom mais une syntaxe ou des fonctionnalités différentes. Pour éviter un conflit, scrbase peut en modifier le nom et la définition. L'option InternalOnly permet de modifier l'exécution conditionnelle des commandes internes en entrant une option sans valeur qui définit les instructions de branchement internes.

Alternativement, l'utilisateur précise les commandes qui ne doivent pas être définies comme valeur, en donnant la priorité au remplacement de "\" par "/".

En principe, les auteurs de packs et de classe ne devraient pas user de cette option. L'utilisateur peut spécifier avec ou sans valeur, soit comme option globale dans \documentclass ou via \PassOptionsTopack.

Exemple :
L'utilisateur ne veut pas que scrbase définisse les commandes \ifVTeX et \ifundefinedorrelax. Pour cette raison, il charge la classe:

```
\documentclass%
      [internalonly=/ifVTeX/ifundefinedorrelax]%
    {foo}
```

Le nom de classe foo est utilisé ici comme un espace réservé. Pour la signification des commandes \ifVTeX, \ifundefinedorrelax et autres instructions conditionnelles, reportez-vous à la section 11.3.

12.2. Caractéristiques des familles et de leurs extensions

Comme déjà mentionné dans l'article 11.1, scrbase utilise le pack keyval pour les clés et leurs valeurs. Néanmoins scrbase étend la fonctionnalité de keyval. Attendu qu'une seule famille détient toutes les clés de keyval, scrbase reconnaît également les membres de cette famille.
Par conséquent, une clé peut être détenue par une famille ou par un ou plusieurs de ses membres. En outre, une valeur peut être attribuée à la clé d'une famille, d'un membre de la famille ou de tous les membres de la famille.

\DefineFamily{*nom de famille*}
\DefineFamilyMember[*membre*]{*nom de famille*}

Pour diverses raisons scrbase doit connaître les membres de la famille. Par conséquent, il est nécessaire de définir une nouvelle famille d'abord avec \DefineFamily qui crée une liste de membres vide. Si la famille existe déjà, il ne se passera rien, ce qui signifie qu'une liste de membres déjà existante ne serait pas remplacée. Ensuite, un nouveau membre peut être ajouté à l'aide de \DefineFamilyMember. Si le membre existe déjà, rien ne se passe, mais l'ajouter à une famille qui n'existe pas provoque un message d'erreur. Si aucun membre n'est spécifié, la valeur par défaut "\@ Currname. \@Currext" est utilisée. Lors du chargement d'une classe ou du pack \@currname et \@currext composent ensemble le nom de fichier de la classe ou du pack.

Théoriquement, il est possible de définir un membre sans nom à l'aide d'un argument optionnel vide de membre de la famille qui correspondrait à la famille elle-même. Il est recommandé de n'utiliser que des lettres et des chiffres pour la famille mais surtout que le premier caractère d'un membre de la famille ne soit ni une lettre , ni un chiffre pour éviter que les membres d'une famille soient les mêmes que les membres d'une autre famille.
scrbase s'attribue l'appartenance à la famille "KOMA" et lui ajoute son fichier "scrbase.sty" comme membre. En principe, les noms de famille "KOMA" et KOMA-Script sont réservés. Pour vos propres packs, il est recommandé d'utiliser le nom global en tant que nom de la famille et le nom des packs individuels en tant que membres.

Par exemple, supposons que vous écrivez un nouveau pack "boucher". Il contient les

packs salami.sty, merguez.sty et boudin.sty. Par conséquent, vous décidez d'utiliser le nom de famille "boucher" et, pour chacun des fichier de pack, vous ajoutez les lignes

```
\DefineFamily{boucher}
\DefineFamilyMember{salami}
\DefineFamilyMember{merguez}
\DefineFamilyMember{boudin}
```

Lors du chargement des trois packs, cela va ajouter les membres "salami.sty", " merguez.sty", et "boudin.sty" à la famille boucher. Après avoir chargé les trois forfaits, les trois membres sont alors définis.

\DefineFamilyKey [membre de la famille]{famille}{key}[par défaut]{action}
\FamilyKeyState
\FamilyKeyStateUnknown
\FamilyKeyStateProcessed
\FamilyKeyStateUnknownValue
\FamilyKeyStateNeedValue

La commande \DefineFamilyKey définit une clé. Si un membre de la famille est donné, la clé devient un attribut de ce membre. Si aucun membre n'est indiqué, le membre ". \@ Currname. \@ Currext" est donné par défaut. Si, plus tard, une valeur est affectée à la clé, l'action est exécutée avec la valeur qui devient argument d'action interne "# 1" et prend cette valeur. Une valeur omise amène l'utilisation de la valeur par défaut. S'il n'existe pas de valeur par défaut, la clé ne peut être utilisée qu'avec une valeur définie.
Cela conduit finalement

```
\DefineFamilyKey[membre]{famille}{key}
                [défaut]{action}
```

à l'appel de

```
\définir@key{membre de la famille}{key}
                [défaut]{action prolongée}
```

où \define@key est déterminée dans le pack keyval (voir [Car99b]). Cependant, répondre à l'appel de \define@key nécessite quelques précautions supplémentaires et l'action est élargie pour inclure ces mesures.

Le succès ou l'échec de l'exécution de l'action sera présenté par \FamilyKeyState à scrbase afin de prendre des mesures supplémentaires si nécessaire et utiliser l'option. par défaut \FamilyKeyState identique à \FamilyKeyStateUnknown. Cela signifie que l'on ne sait pas si oui ou non l'exécution s'est faite correctement et si l'état ne change pas jusqu'à la fin de l'action, scrbase écrira une note dans le fichier journal reprise par \FamilyKeyStateProcessed accepté dans la procédure ultérieure.

L'état \FamilyKeyStateProcessed indique que l'option et l'affectation de la valeur a été achevée et que tout est en ordre. cette condition peut être facilement changée en appelant \FamilyKeyStateProcessed.

L'état \FamilyKeyStateUnknownValue indique que l'option a été bien traitée, mais qu'une valeur doit être affectée, ce qui n'est pas autorisé. Cet état, par exemple,

rapporte qu'il essaye la possibilité d'affecter à twoside une valeur inconnue avec typearea. Le choix de l'Etat se fait simplement en appelant \FamilyKeyStateUnknownValue.

L'état \FamilyKeyStateNeedValue indique que l'option ne peut pas être traitée car une valeur est nécessairement attendue, et l'appel a été fait sans une affectation de valeur. Cet état est défini spontanément quand une option échoue car elle est utilisée sans affectation de valeur, Cependant, en appelant \FamilyKeyStateNeedValue un changement est théoriquement possible.
En outre, des messages d'erreurs supplémentaires peuvent être définis à l'aide de \FamilyKeyState. En général, les quatre états prédéfinis suffisent et devront donc être employés.

Exemple :
Supposons que chacun des trois packs de notre exemple obtienne une clé nommée charcuterie. Lorsqu'il est utilisé, un commutateur doit être fixé à chaque pack en conséquence avec la valeur de l'appel. Le pack salami pourrait par exemple ressembler à ceci:

```
\newif\if@Salami@charcuterie
\DefineFamilyKey{boucher}%
{charcuterie}[true]{%
\expandafter\let\expandafter\if@Salami@charcuterie
\csname if#1\endcsname
\FamilyKeyStateProcessed
}
```

La valeur est donc « true » ou « false » lors de l'appel autorisé. Un test pour des valeurs non valides n'existe pas dans cet exemple. Au lieu de cela, il est toujours fait rapport que l'option a été traitée complètement et correctement. La clé ne sera utilisée que plus tard si l'une des valeurs autorisées doit être affectée à un appel ou sans une affectation de valeur. Dans ce dernier cas la valeur par défaut est attribué « true ».

Les deux autres packs peuvent être définis d'une manière identique. Seul "salami" doit être remplacé par les noms correspondants.

\RelaxFamilyKey[membre]{famille}{clé}

La clé est définie comme attribut d'un membre de la famille, mais cette définition est pratiquement éliminée. La clé ne définit plus ce membre et l'utilisation d'une clé qui n'est pas définie pour ce membre de la famille est également autorisée.

\FamilyProcessOptions[membre]{famille}

En principe, l'extension des familles de clés aux familles et aux membres de la famille est prévue et la clé ou l'affectation de valeur peut être utilisée comme option normale de classe ou de pack. Par conséquent, cette déclaration représente une expansion de l'option \Process * à partir du noyau de LaTeX, (voir [Tea06]), qui traite l'instruction non seulement des options qui ont été définies par \DeclareOption, mais aussi les clés d'un membre spécifié de la famille traitée. Si l'argument optionnel membre n'est pas spécifié, l'option membre \@currext \@Currname sera utilisée.

Les clés de recherche sont définies avant les clés des membres de la famille pour les clés spéciales de fonctions qui ne sont pas associées à un membre de la famille, ou pour une famille dans chaque membre est resté vide.

Exemple :
Si dans l'exemple précédent, un pack est prolongé par la ligne

```
\FamilyProcessOptions{boucher}
```

l'utilisateur peut déjà charger le pack et sélectionner les propriétés des viandes. Si l'option est globale, ce qui est indiqué dans l'argument optionnel \documentclass, la propriété est automatiquement réglée pour les trois packs lorsque les trois packs sont chargés. Les packs traitent toujours les options globales avant les options locales. Lors du traitement des options globales, une valeur d'option inconnue est inscrite dans le fichier journal, ce qui entraîne un message d'erreur, et provoque le renvoi de l'option globale vers les options locales.

On peut interpréter \FamilyProcessOptions comme une extension de \ProcessOption* ou une extension de keyval clé=valeur du mécanisme. En fin de compte, avec l'aide de \FamilyProcessOptions, clé=valeur devient une option.

Lors de l'exécution des options, \FamilyProcessOptions ne doit pas être utilisé. Il est donc interdit dans l'exécution des options pour charger les packs qui auto-utilisent \FamilyProcessOptions.

\FamilyExecuteOptions[membre]{*famille*}{liste d'o*ptions*}

Cette commande est une extension de \ExecuteOptions du noyau LaTeX (voir [Tea06]). Les processus de commande traitent non seulement les options qui sont définies à l'aide de \DeclareOption, mais aussi tous les processus clés des membres d'une famille donnée. Si l'argument optionnel membre n'est pas spécifié, alors \@Currname \@Currext est utilisé.

Une singularité : les clés qui ne sont pas associées à un membre de la famille, mais à une famille dont chaque membre est resté vide sont définies comme clés des membres de la famille.

Exemple :
Supposons que l'option charcuterie doit être définie dans l'exemple précédent par défaut. Les packs n'ont besoin que d'une ligne

```
\FamilyExecuteOptions{boucher}{charcuterie}
```

sera ajoutée

Cette déclaration peut également être utilisée dans d'autres options d'exécution.

\FamilyOptions{*Famille*}{l*iste des options*}

La liste des options prend la forme:

clé=valeur, clé=valeur...

où quand les clés présentent une carence définie de valeur, l'attribution de la valeur

peut également être omise.

Contrairement aux options régulières définies par \DeclareOption, la clé peut être réglée après le chargement d'une classe ou du pack. A cet effet, l'utilisateur utilise \FamilyOptions. Les clés spécifiées de tous les membres de la famille y sont accessibles. S'il existe une clé comme attribut de la famille elle-même, elle est placée en premier. Après cela, les clés de membre suivent dans l'ordre dans lequel les éléments ont été définis. Si une clé donnée n'existe pas, que ce soit pour la famille ou pour un membre de la famille, le pack Scrbase signale une erreur. Cela peut se produire, bien que certains membres possèdent une clé, mais chacun de ces membres signale une erreur sur \FamilyKeyState.

Exemple :
Vous étendez votre projet de boucherie avec salade de saucisses. Si le progiciel est utilisé, tous les packs doivent d'abord produire la charcuterie:

```
\Providespack{salade de saucisse}%
                    [2008/05/06 nonsense package]
\RequirePackage{scrbase}
\DefineFamily{boucher}
\DefineFamilyMember{boucher}
\FamilyProcessOptions{boucher} \relax
\FamilyOptions{boucher}{charcuterie}
```

Si aucun pack de saucisses n'est chargé, un message d'erreur sera émis en raison de l'option "undefined" de charcuterie. Ceci peut être évité par une clé appropriée définie avant la dernière ligne pour le pack lui-même :

```
\DefineFamilyKey{boucher}%
                        {charcuterie}[true]{}%
```

produisant ainsi des packs de saucisses non coupées qui sont chargés après la salade de saucisse. Cela peut aussi changer pour :

```
\AtBeginDocument{%
  \DefineFamilyKey{. salade de saucisse;styl}%
                                {boucher}%
                                {charcuterie}[true]{}%
  }
  \DefineFamilyKey{boucher}%
                                {charcuterie}[true]{}%
  \AtBeginDocument{\FamilyOptions{boucher}
        {charcuterie}{%
  }%
  }%
```

Ainsi, la première option définie avec \begin (document) n'a plus de fonction pour le pack salade de saucisse, à ce moment, \@currname et \@currtext ne sont plus valables et l'argument optionnel de \DefineFamilykey doit être utilisé. Cependant, jusqu'à la redéfinition de l'option, une définition est utilisée par défaut, tandis que le \begin (document) exécute à nouveau l'option pour toute la famille et l'établit aussi à d'autres packs de saucisses.
\FamilyOption{Famille}{option}{liste de valeurs}

En plus des options qui ont des valeurs mutuellement exclusives, il peut y avoir des options avec plusieurs valeurs en même temps. Pour cela, il serait nécessaire lors de l'utilisation de \FamilyOptions, d'assigner à plusieurs reprises, une valeur différente à l'option et ainsi de préciser à plusieurs reprises les options. Vous pouvez aussi attribuer, à une option unique de \FamilyOption, toute une liste de valeurs.

La liste des valeurs est composée d'éléments séparés par des virgules:
{valeur, valeur...}

Dans ce contexte, l'utilisation d'une virgule n'est possible que si les valeurs sont placées entre accolades. La fonctionnalité générale de cette commande est la même que \FamilyOptions.

Exemple :
Le pack de salade de saucisse doit contenir une option par laquelle vous pouvez déterminer d'autres ingrédients. Un commutateur est défini pour chaque ingrédient.

```
\newif\if@salade avec@oignon
\newif\if@salade avec@concombre
\newif\if@salade avec@poivron
\DefineFamilyKey{boucher}{salade supplémentaire}{%
        \csname @salatmit@#1true\endcsname
}
```

Ici les trois ingrédients "oignon", "concombre" et "poivron" ont été définis. Un message d'erreur pour les ingrédients "non définis" n'existe pas.

Pour une salade avec des oignons et des concombres, l'utilisateur peut employer

```
\FamilyOptions{boucher}{%
{Salade supplémentaire=salade, oignon, concombre addendum},
```

ou plus simplement

```
\FamilyOptions{boucher}{%
{ingrédients salade}{oignon,cornichon}
```

ou encore utiliser.

```
{ingredient=oignon,ingredient=cornichon}
```

\AtEndOfFamilyOptions {action}

Parfois, il est utile de retarder l'exécution d'une action qui s'inscrit dans le cadre de l'affectation de valeur à une clé jusqu'à ce que tous les travaux à l'intérieur de \FamilyProcessOptions, \FamilyExecuteOptions, \FamilyOptions ou \FamilyOption soient terminés. Ceci peut être effectué en utilisant une définition de \AtEndOfFamilyOptions. Le rapport d'état de défaillance de l'action n'est pas possible dans ce cas. En outre, la commande ne doit pas être utilisée en dehors d'une définition de l'option.

\FamilyBoolKey[membre]{famille}{clé}{nom de l'interrupteur}
\FamilySetBool{famille}{clé}{nom de l'interrupteur}{valeur}

Dans les exemples précédents les commutateurs ont été utilisés à plusieurs reprises. Dans l'exemple avec l'option charcuterie, il est nécessaire d'indiquer une valeur de l'utilisateur telle que « true » ou « false ». Il n'y aura pas de message d'erreur si l'utilisateur utilise une valeur erronée. Pour cette raison, les interrupteurs booléens sont une caractéristique d'application commune qui permet de paramétrer, avec scrbase, facilement \FamilyBoolKey pour la définition des options de recherche.

Les arguments de la famille, du membre de la famille et de la clé sont les mêmes que ceux utilisés par \DefineFamilyKey.

Le nom de l'argument commutateur est le nom d'un commutateur sans le préfixe \if. Si un commutateur de ce nom n'existe pas déjà, il est automatiquement défini et initialisé à « false » par \FamilyBoolKey qui remplace en interne,\FamilySetBool par \DefineFamilyKey. La valeur par défaut pour une telle option est toujours « true ».
D'autre part, \FamilySetBool comprend l'ajout des valeurs on et off pour allumer et éteindre le commutateur. Si une valeur inconnue est passée, la commande \FamilyUnkownKeyValue est appelée avec la famille des arguments et la clé et la valeur sont reprises dans \FamilyKeyState. Un message d'attribution de valeur inconnue est affiché si nécessaire.

Exemple :
La clé charcuterie doit être définie de manière un peu plus explicite dans les packs de saucisses qui utilisent, en outre, la même clé de sorte que tous les packages, ou aucun d'entre-eux ne produira le mot charcuterie.

```
\FamilyBoolKey{boucher}{charcuterie}%
                                {@charcuterie}
```

Un test avec la charcuterie devrait ressembler à ceci:

```
\si @charcuterie
...
\else
...
\fi
```

Ce serait alors identique dans les trois packs de saucisses. Ainsi, on pourrait définir la propriété "charcuterie" en tant que propriété de principe de la famille:

```
 \@ifundefined{if@charcuterie}{%
   \expandafter\newif\csname if@charcuterie\endcsname
 }{}%
\DefineFamilyKey[]{boucher}{charcuterie}[true]{%
   \FamilySetBool{boucher}{charcuterie}%
                                {@charcuterie}%
                                {#1}%
 }
```

ou plus simplement

```
\FamilyBoolKey[]{boucher}{charcuterie}%
                                {@charcuterie}
```

profitant de la référence secondaire qui s'applique non seulement à \DefineFamilyKey, mais aussi à \FamilyBoolKey. Puisque \FamilyKeyState est déjà défini par \FamilySetBool, l'état de définition de l'option peut être aussi demandé avec \DefineFamilyKey. Donc, vous pourriez dans le premier cas, par exemple, après \FamilySetBool réaliser un test du type:

```
\ifx\FamilyKeyState\FamilyKeyStateProcessed
    ...
\else...
\fi
```

l'utilisation des informations supplémentaires est valable pour \DefineFamilyKey mais aussi pour \FamilyBoolKey, \DefineFamilyKey et pour \FamilySetBool.

A ce stade, selon l'exécution de \FamilySetBool réussie ou non, un test utilisant \ifx est nécessairement fait. Les essais d'expansion tels que \ifstr sont à éviter ici.

```
\FamilyNumericalKey [membre]{famille}{clé}
                    [valeur d'échec]{nom de macro}{liste de valeurs}
\FamilySetNumerical {famille}{clé}
                    {nom de macro}{liste de valeurs}{valeur}
```

Tandis que les commutateurs ne prennent que deux valeurs, une clé admet des valeurs multiples. Par exemple, un alignement peut être à gauche ou ne pas être à gauche, mais aussi à gauche, au centre ou à droite. En interne, une telle distinction est alors paramétrées en utilisant \ifcase. Cette déclaration de TEX attend une valeur numérique. Par conséquent, dans scrbase, on peut attribuer une définition de macro via une clé en utilisant \FamilyNumericalKey.

La liste des valeurs a la forme

{valeur}{d*efinition*},{valeur}{d*efinition*},...

La liste de valeurs ne définit pas uniquement les valeurs autorisées et prises en charge par la clé, mais chacune de ces valeurs si une définition de la commande \macro est donnée. Habituellement il s'agit simplement d'une valeur numérique.

Néanmoins, d'autres contenus sont possibles et permis. Il existe actuellement une restriction, mais cette définition doit être entièrement extensible et doit se développer dans l'attribution en interne.

Exemple :
La saucisse de la salade de saucisse peut être coupée de diverses façons. Il est concevable que les saucisses soient coupées en lanières grossières, en lanières plus fines, ou restent non coupées. Cette information doit être stockée dans la déclaration de salade \coupe.

```
\FamilyNumericalKey{boucher}%
                        {section salade}{section}{%
```

```
    {aucune}{rien},{non}{point},%
    {grossier}{grossier},%
    {fin}{fin}%
}
```

Dans ce cas, aucune coupe par l'utilisateur, ni avec

```
\FamilyOptions {boucher}{salade coupe=aucune}
```

ou ni même avec

```
\FamilyOptions {boucher}{salade coupe=point}
```

Dans les trois cas \coupe ne serait pas définie avec un contenu. Il peut être utile, comme dans cet exemple, de proposer des valeurs multiples à l'utilisateur pour le même résultat.

En outre, il est probable que la coupe ne soit pas éditée, mais évaluée plus tard. Dans ce cas, une définition textuelle ne serait pas pratique, mais si l'on définit la clé ainsi, elle pourra être utilisée par \FamilyNumericalKey \DefineFamilyKey avec la commande \FamilySetNumerical.

```
\FamilyNumericalKey{boucher}%
        {section de salade}{section}{%
        {aucune}{0},{non}{0},{point}{0}%
        {gros}{1},%
        {fin}{2}%
        }
```

et vous pourrez ensuite distinguer facilement une condition semblable à la suivante qui sera utilisée sous la forme :

```
\ifcase\coupe
        % pas de coupe
\or
        % grossièrement haché
\else
        % haché finement
\fi
```

Lors de l'appel, la réussite ou l'échec est interprété automatiquement par \FamilyKeyState.

En interne de \FamilyNumericalKey, \DefineFamilyKey est alors utilisé avec l'instruction \FamilySetNumerical. Si une valeur inconnue est assignée à une clé, \FamilySetNumerical appellera \FamilyUnkownKeyValue avec les arguments de la famille, clé et valeur.

Cela conduit à une signalisation d'erreur dans \FamilyKeyState utilisant la condition \FamilyKeyStateUnkownKeyValue et, par exemple, lorsqu'il est utilisé comme une option locale à un message d'erreur. De même, lorsque vous appelez \FamilySetNumerical le succès via \FamilyKeyStateProcessed est signalé dans \FamilyKeyState.

```
\FamilyCounterKey[Membre]{Familie}{clé}
        [valeur par défaut]{LaTeX-compteur}
\FamilySetCounter{Famille}{clé}{LaTeX-compteur}{valeur}
```

Alors que dans \FamilyNumericalKey une macro est réglé sur une valeur numérique correspondant à cause d'une valeur symbolique, il existe naturellement des cas dans lesquels une clé représente directement un compteur de LaTeX avec une valeur numérique à attribuer immédiatement.

C'est le but de la commande interne \FamilyCounterKey puis \FamilySetCounter est appelé. Voici quelques vérifications de base la valeur des arguments plutôt que de déterminer si cet argument pour une mission vient à un compteur en question. La répartition n'aura lieu que si les tests réussissent. Cependant, vous n'aurez pas tous les défauts qui sont détectés, de sorte que l'attribution incorrecte peut aussi conduire à un message d'erreur de TeX lui-même. Les erreurs détectées sont signalées par \FamilyKeyStateUnknownValue.

Si aucune valeur n'est passée, la valeur par défaut est utilisée. Si aucune valeur par défaut n'est spécifiée, la clé peut être utilisée plus tard uniquement avec le transfert de la valeur.

```
\FamilyCounterMacroKey[Membre]{Famille}{clé}
        [valeur par défaut]{macro}
\FamilySetCounterMacro{Famille}{clé}{macro}{valeur}
```

Ces deux déclarations sont différentes de \FamilyCounterKey précédemment déclarée et \FamilySetCounter seulement le fait que pas un compteur de LaTeX n'est réglé à une valeur, mais une macro est définie par cette valeur.

```
\FamilyLengthKey[Membre]{Famille}{clé}
                    [valeur par défaut]{Longueur}
\FamilySetLength{Famille}{clé}
                    {longueur}{valeur}
\FamilyLengthMacroKey[Membre]{Famille}{clé}
                    [valeur par défaut]{Macro}
\FamilySetLengthMacro{Famille}{clé}
                    {Macro}{valeur}
```

À propos de \FamilyLengthKey une clé peut être définie représentant une longueur. Il importe peu de savoir s'il s'agit de longueur LaTeX, de distance ou d'extension TeX qui est utilisée. En interne, la longueur est définie par \FamilySetLength. Mieux vaut vérifier l'argument de la valeur de base plutôt que déterminer si cet argument est adapté pour une affectation à une longueur.

La répartition n'a lieu que si les tests réussissent. Cependant, tous les défauts ne sont pas détectés et une attribution incorrecte peut conduire à un message d'erreur TeX. Les erreurs sont signalées par \FamilyKeyStateUnknownValue.

Si aucune valeur n'est passée, la valeur par défaut est utilisée à la place. Si la valeur par défaut manque, la clé peut être utilisée plus tard uniquement avec un transfert de valeur.

Si vous utilisez les commandes \FamilyLengthMacroKey et \FamilySetLengthMacro

le stockage de la valeur n'a pas lieu dans une \longueur, mais dans une \macro.

\FamilyStringKey[membre]{famille}{clé}
[valeur par défaut]{macro}

Ceci définit une clé qui peut prendre n'importe quelle valeur. La valeur sera stockée dans la macro spécifiée. s'il n'existe aucun argument optionnel pour la valeur par défaut, le code \FamilyStringKey correspond à :

\DefineFamilyKey[membre]{famillle}{clé}
{\defMakro {#1}}

Elle indique, en outre, si l'argument optionnel pour la valeur par défaut est omis avec l'appel de

\DefineFamilyKey[membre]{famille}{clé}
[valeur par défaut]
{\defMakro {#1}\FamilyKeyStateProcessed}.

Si la commande n'est pas définie, elle est définie comme une macro vide.

Exemple :
Par défaut, 250 g de salade de saucisses doit être produite. Ce montant doit être simple à modifier à l'aide d'une option. A cet effet, la quantité doit être créée par la macro enregistrée \poids de la salade. L'option qui change le poids doit pouvoir également gérer le poids de la salade chaude:

\newcommand* {\poidsdesalade}{250g}
\FamilyStringKey {boucher}%
{poidsdesalade}[250g]{\poidsdesalade}

Pour revenir au poids par défaut après l'avoir modifié, l'utilisateur peut utiliser simplement l'option sans poids:

\FamilyOptions{boucher}{poidsdesalade}

Ceci est possible parce que la quantité standard a été fixée dans la définition comme valeur par défaut.

Dans ce cas, il n'y a pas de valeur inconnue car toutes les valeurs sont simplement utilisées pour une définition de macro. Cependant, dans cette attribution, aucune valeur ne peut être incluse dans la clé.

\FamilyUnkownKeyValue{*Famille*}{clé}{valeur}{liste des valeurs}

La commande \FamilyUnkownKeyValue lance par \FamilyKeyState un message d'erreur sur les valeurs inconnues ou non autorisées attribuées à une clé connue. La liste des valeurs est une liste de valeurs autorisées, séparées par des virgules, sous la forme:

‘valeur’, ‘valeur’...

Actuellement, les valeurs ne sont pas évaluées par la liste de scrbase.

Exemple :
Désormais, les coupes n'indiqueront plus si la coupe doit être grossière ou fine. Un préréglage par défaut désigne le type de coupe à utiliser, même s'il n'est pas précisé comment les coupes doivent couper.

```
\@ Ifundefined{if@coupe fine}{%
\expandafter
\newif \csname if@coupe fine\endcsname}{}%
\@ Ifundefined {if@coupe}{%
\expandafter
\newif \csname {if@tranché}\endcsname}{}
\DefineFamilyKey{boucher}%
                        {coupe}[true]{%
\FamilySetBool{boucher}{coupe}%
                                    {@coupe}%
                                    {#1}%
\Ifx \FamilyKeyState\FamilyKeyStateProcessed
\@{coupe fine}{false}
\else
\ifstr{#1}{fin}{%
\@{jambon}{true}
\@{coupe fine}{true}
\FamilyKeyStateProcessed
}{}%
\fi
}%
```

Nous réglons d'abord, à l'aide de \FamilySetBool, le commutateur booléen pour les coupes, nous testons ensuite si \FamilyKeyState signale la réussite de la commande à l'état \FamilyKeyStateProcessed. Si c'est un succès la coupe fine est activée. Sinon, il est vérifié si une valeur valide a été adoptée pour le commutateur booléen.

Si aucune des deux commandes n'a été adoptée, elles seront réglées et activées à l'aide de \FamilyKeyStateProcessed. Si le test de charge échoue également, l'échec de \FamilySetBool est signalé et enregistré à la fin de l'exécution. La déclaration utilisée dans les tests \ifstr est expliquée section 11.3.

\FamilyElseValues

Dans les versions antérieures de scrbase vous pouviez autoriser d'autres valeurs en utilisant la commande \FamilyElseValues pour un traitement et les stocker sous la forme

'Valeur', 'valeur'...

par \FamilyUnknowValue qui pouvait diffuser alors un message d'erreur.

Depuis la version v3.12 \FamilyUnknowValue ne signale plus de message d'erreur, et utilise \FamilyKeyState.

En conséquences, l'utilisation de \famille ElseValue est dépassée mais elle est détectée et elle conduit à un appel d'ajustement du code en conséquence.

12.3. Exécution conditionnelle de scrbase

Le pack de scrbase fournit plusieurs commandes d'exécution optatives différentes de la syntaxe de TEX, construites telles que :

```
\iftrue
...
\else
...
\fi
```

dont la syntaxe est utilisée avec des arguments connus par les commandes LaTeX comme \IfFileExists, \@ifundefined, \@ifpackloaded et bien d'autres.

Néanmoins, certains auteurs de packs préfèrent utiliser la syntaxe de TEX, même pour les utilisateurs de l'interface LaTeX, ce qui pourrait inévitablement conduire à des conflits de noms avec scrbase, la probabilité d'utilisation d'instruc-tions identiques étant très forte, même avec une sémantique analogue, et poser problème en raison de la syntaxe différente.

En fait, les conditionnelles de scrbase sont très basiques :

Scrbase définit par conséquent certaines de ces instructions, d'abord en tant que commandes internes avec le préfixe \scr@, les commandes sans ce préfixe peuvent être définies ultérieurement par l'utilisateur, ce qui peut être évité en utilisant l'option InternalOnly (voir la section 11.1).

Les auteurs de packs et de classes devraient utiliser les commandes internes de KOMA-Script, et, pour être complets, les instructions d'utilisation qui les accompagnent.

\scr@ifundefinedorrelax{*nom*}{instructions}{autres instructions}
\ifundefinedorrelax{*nom*}{puis instructions}{autres instructions}

Cette commande fonctionne essentiellement comme \@ifundefined du noyau LaTeX (voir [BCJ 05]). Les instructions sont exécutées lorsque le nom de la commande est indéfini ou actuellement \relax. Contrairement à \@ifundefined, ce nom, s'il était précédemment indéfini, n'est pas défini \relax par défaut.
En outre, l'utilisation de ε-TEX n'a pas d'effet mémoire.

\scr@ifpdftex{alors des instructions}{et d'autre part}
\ifpdftex{puis la partie}{et d'autre part}

Si pdfTEX a été utilisé, les instructions sont exécutées sinon par défaut l'instruction else. Il n'est pas important de savoir si un fichier PDF a été effectivement généré ou pas, information rarement vraiment utile. Mais en général, vous devriez tester d'avantage la commande souhaitée.

\scr@ifVTeX{alors l'instruction}{ou l'instruction else}
\ifVTeX {alors l'instruction}{ou l'instruction else}

Si VTEX a été utilisé, les instructions seront ensuite exécutées sinon par défaut l'instruction else. Ce test est rarement utile. Par contre, vous devriez tester d'avantage l'instruction souhaitée. (voir précédente observation)

\scr@ifpdfoutput {alors l'instruction}{ou l'instruction else}
\ifpdfoutput{alors l'instruction}{ou l'instruction else}

Si un fichier PDF est créé, alors les instructions sont exécutées, sinon la partie else. Il importe peu que le fichier PDF soit généré à l'aide de pdfTEX, de VTEX ou d'un autre moteur de TEX.

\scr@ifpsoutput{alors des instructions}{et d'autre part}
\ifpsoutput{puis la partie}{et d'autres Instructions}

Si un fichier PostScript est créé, les instructions sont exécutées, sinon les instructions else. VTEX peut directement générer un PostScript, qui sera, ici, reconnu. Si VTEX n'est pas utilisé, on peut définir un bouton avec \if@dvips, et la décision dépendra de ce commutateur fourni par KOMA-Script dans le pack typearea.

\scr@ifdvioutput{des instructions}{et d'autre part}
\ifdvioutput{alors des instructions}{et puis aussi}

Lors de la génération d'un fichier DVI, les instructions sont exécutées, sinon l'instruction else. Un fichier DVI est généré quand aucune sortie directe vers un fichier PDF ou PostScript ne peut être reconnue.

\ifnotundefined{nom}{alors des instructions}{puis la partie}

ε-TEX sera utilisé pour tester si une commande avec un nom donné a déjà été définie ou pas. Si la commande est définie, les instructions seront ensuite exécutées. Il n'existe pas d'option interne à cette déclaration.

\ifstr{chaîne}{chaîne}{alors des instructions}{sinon l'instruction else}

Les deux arguments de chaîne sont développés et ensuite comparés. Si les extensions sont les mêmes, les instructions sont réalisées, sinon l'instruction else est exécutée. La commande n'est pas complètement extensible et il n'existe pas d'option interne à cette déclaration.

\ifstrstart{chaîne}{chaîne}{alors des instructions}{ou l'instruction else}

Les deux arguments de chaîne sont développés et ensuite comparés. Si le côté des espaces blancs de la première chaîne commence la seconde, alors les instructions seront réalisées, sinon les autres instructions seront exécutées. Les expressions invalides provoquent un message d'erreur.
La commande n'est pas complètement extensible et il n'existe pas d'instruction interne correspondante.

\ifisdimen{*Code*}{alors des instructions}{ou l'instruction else}

Si le résultat de l'expansion du code dans un \dimen, est reconnu comme valeur de longueur de TEX, alors les instructions seront réalisées, sinon l'instruction else sera exécutée. Les expressions invalides provoquent un message d'erreur. La commande n'est pas complètement extensible et il n'existe pas d'instruction interne correspondante.

\ifisdimension{*Code*}{alors des instructions}{l'instruction else}

Si le résultat de l'expansion du code dans un \dimexpr est reconnu comme expression d'une longueur TeX, alors les instructions seront réalisées, sinon l'instruction else sera exécutée. Les expressions invalides provoquent un message d'erreur. La commande n'est pas complètement extensible et il n'existe pas d'instruction interne correspondante.

\ifisdimexpr{Code}{alors l'instruction}{ou l'instruction else}

Si le résultat de l'expansion du code dans un \skip, est reconnu comme registre à distance de TEX, alors les instructions seront réalisées, sinon l'instruction else est exécutée. Les expressions invalides provoquent un message d'erreur. La commande n'est pas complètement extensible et il n'existe pas d'instruction interne correspondante.

\ifisskip{Code}{{alors l'instruction}{ou l'instruction else}

Quand l'expression du code \glue correspond à la syntaxe de la valeur d'une distance, la première partie est exécutée, sinon la partie de l'instruction else. Une commande invalide provoque un message d'erreur. La déclaration n'est pas entièrement extensible et il n'en existe pas de variation interne.

\ifisglue{*code*}{alors l'instruction}{ou l'instruction else}

Si le code de l'expansion donne un résultat dans \glueexpr reconnu comme expression de distance par TEX, les instructions seront réalisées, sinon l'instruction else sera exécutée. Les expressions invalides provoquent un message d'erreur. La commande n'est pas complètement extensible et il n'existe pas d'instruction interne correspondante.

\ifisglueexpr{*Code*}{alors l'instruction}{ou l'instruction else}

Si le code de l'expansion dans \count donne un résultat reconnu comme registre de compteur de TeX, les instructions seront exécutées, sinon l'instruction else sera appliquée. La commande n'est pas complètement extensible et il n'existe pas de commande interne correspondante. Pour le test avec des compteurs de LaTeX, voir \ifiscounter.

\ifiscount{code}{alors l'instruction}{ou l'instruction else}

Lorsque le résultats de l'expansion du code correspondant à la syntaxe de la valeur d'un compteur est un nombre entier positif ou négatif, la commande est effectuée, sinon l'instruction else sera appliquée. La déclaration n'est pas entièrement extensible et il n'existe pas de commande interne correspondante.

\ifisnumber{code}{les instructions}{ou l'instruction else}

Si le code dans un \numexpr, est le résultat d'une expression de nombre TEX, alors les instructions sont exécutées, sinon l'instruction else est utilisée. Les expressions défectueuses mènent à un message d'erreur. La version n'est pas entièrement extensible et il n'existe pas de commande interne correspondante.

\ifisnumexpr{*Code*}{les instructions}{ou l'instruction else}

Si le compteur est identifié comme un compteur LaTeX les instructions sont exécutées, sinon l'instruction else est utilisée. La version n'est pas entièrement extensible et il n'existe pas de commande interne analogue.

\ifiscounter{compteur}{les instructions}{ou l'instruction else}

Ici, aucun nombre n'est comparé et les instructions sont exécutées lorsque la chaîne ne comporte que des chiffres. A défaut, l'instruction else est appliquée. Il n'existe pas de commande interne correspondante à la déclaration.

\ifnumber{*chaîne*}{les instructions}{ou l'instruction else}

Ici aucun décompte n'est comparé. La partie fonctionne quand l'expansion de la chaîne se compose uniquement de chiffres, sinon, la partie else sera utilisée. Il n'existe pas de variation interne de cette déclaration,

\ifdimen{*chaîne*}{les instructions}{ou l'instruction else}

Ici, aucune longueur n'est comparée. La pièce est exécutée lorsque l'extension de la chaîne est une longueur valide avec une unité de longueur valable. Autrement, la partie else sera utilisée. Il n'existe pas de variation interne de cette déclaration,

\if@document instructions \else instructions *fi*

Cette branche existe délibérément comme une déclaration interne, \if@document correspond au préambule du document, après \begin{document} se trouvent les conditionnelles \iffalse et \iftrue.
Les classes et les auteurs peuvent utiliser ces instructions souvent très utiles quand existe une différence de comportement entre préambule et corps du document. Il est important de noter que cette branche figure dans la syntaxe de TeX et non dans celle de LaTeX pour cette déclaration!

12.4. Identificateur de la langue

Normalement, il faut modifier ou définir les termes dépendant de la langue telle que \captions en français de manière à ce que, en plus des conditions disponibles, les nouveaux termes soient définis ou redéfinis, ce qui parait d'autant plus difficile que certains packs comme german ou ngerman redéfinissent ces paramètres quand ils sont chargés, en supprimant les modifications apportées précédemment.

Il paraît donc logique de retarder les changements avec \AtBeginDocument jusqu'à \begin{document}, le chargement du pack est ensuite terminé. L'utilisateur peut ainsi redéfinir les termes dépendant de la langue après begin\{document}, ou ne pas les mettre du tout dans le préambule.

Un autre inconvénient est que certains packs supplémentaires dépendants aussi de la langue définissent les termes dans \captionsSprache, alors que d'autres utilisent \extrasSprache.
Ainsi, l'utilisateur doit être très exactement informé afin de compléter de la bonne manière une instruction correcte.

Le pack scrbase fournit quelques commandes supplémentaires pour ce faire.

\defcaptionname{liste des langages}{terme}{definition} \providecaptionname{liste des langages}{terme}{definition} \newcaptionname{liste des langages}{terme}{definition} \renewcaptionname{liste des langages}{terme}{definition} \defcaptionname*{liste des langages}{terme}{definition} \providecaptionname*{liste des langages}{terme}{definition} \newcaptionname*{liste des langages}{terme}{definition} \renewcaptionname*{liste des langages}{terme}{definition}

L'utilisation d'une commande de synthèse permet d'assigner une définition en fonction de la langue particulière à un terme. Plusieurs langues séparées par une virgule peuvent être spécifiées dans une liste de langages.

Le terme est toujours une macro. Les commandes varient selon qu'une langue donnée ou qu'un terme dans une langue donnée sont déjà définis ou non au moment où la commande est appelée.

Si une langue n'est pas définie, \providecaptionname ne fait rien d'autre que d'écrire un message dans le fichier journal. Cela ne se produit qu'une seule fois pour chaque langue. Si une langue est définie, mais sans concept correspondant, elle sera définie en utilisant le contenu spécifié. Le terme ne sera pas redéfini si la langue a déjà examiné une définition; et un message approprié sera écrit, à la place, dans le fichier journal.

La commande \newcaptionname a un comportement légèrement différent. Si une langue n'est pas encore définie, une instruction correspondante sera créée et un message écrit dans le fichier journal. Pour la langue USenglish, par exemple, ce serait la commande \captionsUSenglish ou \captionsngerman pour la langue ngerman. Si un terme n'est pas encore disponible dans cette langue, il est défini par le contenu souhaité. S'il existe déjà dans une langue, cela se traduira par un message d'erreur.

La commande \renewcaptionname se comporte à nouveau différemment. Si la langue n'est pas définie, un message d'erreur est émis. Toutefois, si la langue est définie et que le terme est inconnu dans cette langue, un message d'erreur est également délivré. Sinon, le terme de la langue sera redéfini selon la définition avec le contenu souhaité.

KOMA-Script emploie \providecaptionname pour définir les commandes dans la section 21.4.

Exemple :
Si vous préférez « fig. » au lieu de « figure » en frenchb, vous pouvez obtenir cette aide:

```
\renewcaptionname{frenchb}{\figurename}{fig.}
```

Si vous désirez le même changement non seulement dans ngerman mais également dans nswissgerman, ou encore dans Österreichisch et Schweizer Deutsch[1], vous ne avez pas besoin d'un supplément:

1 autrichien et suisse allemand

```
\renewcaptionname{nswissgerman}{\figurename}{fig.}
\renewcaptionname{UKenglish}{\figurename}{fig.}
```

et la liste des langues peut être facilement étendue:

```
\renewcaptionname{ngerman,nswissgerman}{\figurename}{fig.}
\renewcaptionname{USenglish,UKenglish}{\figurename}{fig.}
```

Depuis KOMA-Script 3.12, il n'est plus nécessaire de retarder la définition ou la redéfinition utilisant \AtBeginDocument à \begin {document}. Cela est fait par scrbase même appelée dans le préambule.

En outre scrbase vérifie maintenant également si une nouvelle définition des termes existe dans \captionslanguage ou dans \extras*language*. Les nouvelles versions étoilées des commandes utilisent essentiellement les langues supplémentaires dont les définitions se trouvent généralement après l'application \captionsSprache

Bien que la redéfinition des identifiants spécifiques à la langue du pack fonctionne généralement bien avec hyperref, pensez à utiliser la langue supplémentaire \extraslanguage.
Les classes sont habituellement définies en langue anglaise et les packs sont affichés et décrits dans le tableau 12.1.

Tableau 12.1: Vue d'ensemble de termes habituels dépendant du pack de langue

\abstractname	\rubrique pour le résumé des noms abstraits
\alsoname	\autre appellation pour "voir aussi" dans les références croisées supplémentaires
\appendixname	\titre "annexe" de la tête de chapitre d'une pièce jointe
\bibname	\titre de la bibliographie
\ccname	\préfixe ccname de la rubrique « liste de distribution d'une lettre »
\chaptername	\nom du «chapitre» dans le titre d'un chapitre
\contentsname	\nom de rubrique de la table des matières
\enclname	\préfixes de rubriques incluses dans une lettre
\figurename	\nom de la rubrique signature
\glossaryname	\nom de la rubrique glossaire
\headtoname	\nom de l'en-tête pour une en-tête de lettre
\indexname	\index des mots-clés
\listfigurename	\liste des figures
\headtoname	\en tête des pages de lettres
\indexname	\indice rubrique nom de l'index
\listfigurename	\liste des figures
\listtablename	\liste des tables
\pagename	\page dans la pagination de lettres
\partname	\«partie» dans le titre d'une partie
\prefacename	\rubrique de la préface
\proofname	\nom du préfixe rubrique preuves mathématiques
\refname	\liste des références
\seename	\"voir" les références croisées de l'indice
\tablename	\nom des légendes de table en préfixe de la rubrique table

12.5. Identification de KOMA-Script

Le pack scrbase peut être utilisé indépendamment de KOMA-Script avec d'autres pa-ckages et classes. Néanmoins, c'est un pack KOMA-Script , et pour cette raison, il fournit des commandes pour identifier KOMA-Script et pour s'identifier comme un pack KOMA-Script

\KOMAScript
Cette commande définit uniquement le mot "KOMA-Script " avec la police sans-serif et une fixation des parties en capitales dans l'ensemble. Tous les packs et les classes de KOMA - classes spécifient cette commande, si elle n'a pas déjà été précisée. La définition est donnée par robust utilisant \DeclareRobustCommand.

\KOMAScriptVersion
KOMA-Script définit la version principale de KOMA-Script dans cette commande. Elle a la forme de "version actualisée KOMA-Script ". Cette version principale est la même pour tous les forfaits et toutes les classes de KOMA-Script. Pour cette raison, il peut aussi être appelé après le chargement de scrbase. Par exemple, ce document a été fait en utilisant KOMA-Script "19/12/2013 v3.12 KOMA-Script ".

12.6. Extensions du noyau LaTeX

Dans certains cas, le noyau de LaTeX fournit lui-même des instructions qui sont parfois incomplètes et des commandes similaires seraient souvent utiles. Certaines sont fournies par scrbase pour les auteurs de packs et de classes.

\ClassInfoNoLine{className}{informations}
\packInfoNoLine{nomdupack}{informations}

Le noyau de LaTeX fournit à l'auteur de packs et de classes des commandes telles que \packInfo et \ClassInfo pour écrire des informations avec le numéro de la ligne courante dans le fichier journal.

Il propose \packWarning et \ClassWarning, qui émettent des avertissements avec le numéro de la ligne actuelle.

Les commandes \packWarningNoLine et \ClassWarningNoLine émettent des avertissements sans numéro de ligne.

Les commandes évidentes pour écrire des informations sans numéro de ligne dans le fichier journal, \ClassInfoNoLine et \packInfoNoLine, manquent mais sont fournies par scrbase.

\l@addto@macro{*commande*}{*extension*}

Le noyau de LaTeX fournit une commande interne \g@addto@macro qui permet d'étendre la définition d'une macro-instruction par l'extension globale du code. Cela fonctionne, dans ce formulaire, uniquement pour les macros sans arguments. Pourtant, on pourrait avoir besoin de cette déclaration dans certains cas, sous une forme qui fonctionne globalement. Elle est alors fournie par scrbase sous la forme de \l@addto@macro. Une alternative peut être l'utilisation du pack etoolbox qui offre un certain nombre de ces instructions à des fins différentes (voir [Leh11]).

12.7. Extensions des compétences mathématiques de ε-TeX

ε-TeX qui est utilisé par LaTeX et nécessairement par KOMA-Script , placé avant \numexpr a sa fonctionnalité étendue pour le calcul de l'arithmétique simple avec des compteurs TeX et entiers. Les quatre opérations et les supports arithmétiques de base sont pris en charge et les arrondis sont corrects en division, mais des opérateurs supplémentaires seraient parfois utiles.

\XdivY{*dividende*}{*diviseur*}
\XmodY{*dividende*}{*diviseur*}

La commande \XdivY renvoie la valeur du quotient entier, la commande \XmodY donne la valeur du reste. Ce genre de division est défini par l'équation:

$$dividende = diviseur \cdot quotient\ entier + reste$$

selon lequel le dividende et le reste sont des entiers, le reste est supérieur ou égal au diviseur, et le diviseur est un nombre naturel supérieur à 0.

La valeur peut être utilisée pour l'affectation à un compteur ou directement dans \numexpr. La valeur de sortie à partir de chiffres arabes doit être préfixée par \the.