

Royaume du Maroc



Ministère de l'Education Nationale,
du Préscolaire et des Sports

Manuel de l'utilisateur

Fichiers de style \LaTeX

pour LA PRODUCTION DES LIVRETS DES PROGRAMMES
DES CLASSES PRÉPARATOIRES

version 1.0beta

Manuel de l'utilisateur

Ce manuel est destiné à tous les intervenants dans la chaîne de production des livrets des programmes des CPGE, les rédacteurs, comme ceux qui ont une responsabilité de relecture ou de mise en forme finale. L'objectif est d'harmoniser l'apparence des programmes des différentes matières et de fournir une architecture solide et durable.

L'implémentation a été réalisée selon l'un des principes fondamentaux du système \LaTeX : la séparation entre sémantique et éléments de présentation. Les programmes sont appelés à changer donc la qualité et la clarté des fichiers sources est une condition fondamentale pour pouvoir poursuivre son évolution sans avoir besoin à chaque fois de réinventer la roue. La tâche des rédacteurs de programmes sera la structuration logique du contenu sans avoir à se soucier des aspects de la présentation. Cette dernière sera la responsabilité des fichiers style du projet et de l'équipe de production.

Trop long ?

Ce manuel est long, très long. Il reflète néanmoins le soin apporté à l'élaboration du projet. À mi-chemin entre manuel technique et tutoriel, il ne s'interdit pas de déborder sur des thèmes qui ne sont pas en rapport direct avec son sujet mais qui aident à expliquer les choix qui ont été faits. Dans les versions précédentes il faisait uniquement office de manuel technique. À l'utilisation, les intervenants, à cause des lacunes ou de la mauvaise documentation, se sont sentis obligés d'ajouter des fonctionnalités pour contourner les déficiences apparentes du projet. Maleureusement cela entraient parfois en concurrence, ou en contradiction, avec les fichiers styles. C'est ce qui explique le changement de paradigme dans la rédaction de ce manuel. L'équipe de développement espère que celui-ci contribuera à enrichir leurs expérience avec le système \LaTeX en général et non seulement vis à vis du projet.

Une version plus courte de ce document est préparée à partir du même matériel. Elle se limite à la liste des commandes et de leurs options avec leurs descriptions. Celle-ci servira de manuel de référence au sens plus strict une fois l'utilisateur familiarisé avec les concepts généraux décrit dans cette version.

Niveau exigible

pour bien utiliser les fichiers du projet il suffit d'être familier avec le système \LaTeX en général. Néanmoins une consultation de ce manuel chaque fois que c'est nécessaire peut s'avérer indispensable

pour rester en conformité avec les exigences du projet.

texdoc, mon précieux...

Si vous avez fait une installation complète de votre distribution \LaTeX alors vous avez plusieurs gigaoctets de documentation sur votre système. Cela inclut les documentations de pratiquement tous les packages, des documentations sur \TeX et \LaTeX eux mêmes, certains magazines libre de droit...

Les deux distributions majeures sont accompagnées d'un utilitaire en ligne de commande : texdoc. Ouvrez votre console et tapez par exemple

```
texdoc source2e
```

et vous serez surpris de voir tout le code source commenté et catégorisé de \LaTeX (\LaTeX est entièrement écrit en langage \TeX et non en langage compilé). Un autre document intéressant pour les utilisateurs avancés s'obtient avec

```
texdoc texbytopic
```

Pour une utilisation normale, vous pouvez consulter la documentation d'un package en donnant son nom en argument à texdoc. Par exemple

```
texdoc geometry
```

I Historique des versions

Version 1.0beta

Cette mise à jour est une version majeur des fichiers style du projet. Elle essaye de combler les lacunes des versions précédentes et prend encore mieux en considération *l'évolutivité du projet*. Les programmes sont susceptibles d'être mis à jour et les intervenants peuvent changer. Pour les programmes, il faut que les fichiers sources soient les plus propres et les plus clairs possibles d'où la nécessité d'outils auto-informatifs qui reflètent la sémantique du contenu sans trop encombrer la syntaxe. Pour les utilisateurs, il faut que la prise en main soit la plus naturelle possible ce qui passe par la définition de commandes ergonomiques et surtout par une documentation accessible et bien rédigée.

Les principaux ajouts sont :

- les livrets sont désormais produits par défaut en 11pt au format 19cmx24cm (au lieu de 17cmx24cm qui est mal adapté à du contenu en 2 colonnes). Le fichier de classe supporte aussi une option appelée `a4` qui produit le livret au format A4 pour une impression à domicile. Rappelons qu'il est possible de changer à volonté le format avec les options `width` et `height` de la classe ;
- un restylage très léger des titres ;
- étant le principal outil des versions précédente, l'environnement bicolonne a été enrichi d'une interface de personnalisation de type clé/valeur. Il est maintenant possible d'altérer de façon simple la largeur des colonnes, les sauts verticaux à l'intérieur et à l'extérieur de l'environnement, l'activation d'un filet de séparation... Il est aussi possible d'enregistrer des configurations pour ensuite les utiliser elles mêmes comme options.

- une version plus générale (encore en phase de test) de l'environnement bicolonne pour pouvoir simuler des tableaux avec plus que deux colonnes ;
- les listes (`enumerate`, `itemize`...) se comportent mieux dans l'environnement multicolonne ou dans un tableau. Des outils pour régler leurs propriétés et pour créer de nouvelles structures plus ergonomiques ont été ajoutés ;
- des mécanismes plus simples que ceux standards (de base il n'y en a aucun) pour changer le style des titres et de leur numérotation. Il va de soit que ceux-ci devraient être utilisés avec beaucoup de précaution. Ils sont en fait destinés à l'équipe de production ;
- un mécanisme pour créer à la volée et de manière simple des commandes pour insérer des titres typographiquement corrects et avec compteurs intégrés ;
- la possibilité de produire à partir du même matériel, des documents plus adaptés à une consultation sur écran (tablette numérique ou smartphone).
- une meilleure gestion des erreurs ;
- l'ajout de deux nouveaux fichiers styles `lvrtbase` et `lvrtmath` qui apportent de nouvelles fonctionnalités pour un usage interne par les autres fichiers style et aussi directement dans les fichiers de bases ;
- ce manuel a été considérablement étoffé pour qu'il serve mieux son propos : garantir la pérennité des fichiers du projet.

Cette version est ainsi munie, pour les situations non prévues, d'outils qui permettent d'étendre les fonctionnalités des fichiers style sans nuire à l'harmonie globale des documents.

Un investissement rentable

Les utilisateurs intéressés pourront utiliser toutes les nouvelles fonctionnalités en dehors du projet. Outre l'implémentation spécifique et plus pratique des listes et des commandes de titrage, l'environnement multicolonne peut avoir des applications intéressantes comme maintenir un cahier de texte ou rédiger un rapport d'inspection. Voir la dernière section de ce document pour une exploration de ces possibilités.

Vieux bogue

Depuis la dernière version de ce kit (plus de dix ans maintenant) le package `paracol` a gagné en maturité. Apparemment le bogue des ruptures des page ne se produit plus.

Version 0.61

Plus de soins porté à la présentation de certains aspects des documents, notamment les tables de matières et les couvertures.

Le recours à certaines extensions a été supprimé (notamment `framed`). Le fichier `gtdnames.def` (qui définissait plus de 100 commandes \LaTeX , toutes des raccourcis de noms de savants... fâcheuse maladresse) a été supprimé du projet. Les fichiers de bases qui l'utilisait ont été mis à jours pour utiliser la commande `\nom`.

Les fichiers du projet ont subi une phase préliminaire de débogage. Certains bogues subsistent.

Un bogue particulièrement gênant, et dont l'origine est liée à l'extension `paracol`, survient parfois lorsqu'une page doit commencer par un titre en mode unicolonne. Il peut arriver que la page précédente disparaisse complètement.

Symptôme On peut détecter si une telle erreur s'est produite en comparant le numéro de la dernière page avec le nombre de pages que comporte réellement le document PDF.

Diagnostic On peut ensuite localiser rapidement la page où s'est produit le problème, en procédant par dichotomie sur les numéros de page. La page qui suit la page manquante commence souvent par un titre (`\section` ou `\subsection`). Le titre est alors écrit en couleur normale (noire) au lieu de respecter le style général du document.

Correctif remplacer la commande `\section{<titre>}` qui commence la page qui suit la page manquante par `\bprogram[\section{<titre>}]`. Noter qu'il n'est pas nécessaire d'initialiser un autre bloc `bprogram` dans le cas où un paragraphe du programme suit immédiatement le titre. La commande `\bprogram` peut être remplacée par `\bcomment` si c'est un commentaire qui suit le titre. Faire de même s'il s'agit plutôt d'une commande `\subsection{<sous-titre>}`.

Une solution globale serait de toujours insérer les titres par le biais de la commande `\bprogram` ou `\bcomment`, dans le cas où on doit fermer l'environnement multicolonne juste pour insérer un titre et y revenir immédiatement après.

Les commandes `\Section`, `\subSection` et `\subsubSection` ont été ajoutées pour faciliter l'adoption de cette dernière démarche pour un fichier déjà traité. Chacune de ces commandes remplace la commande sans le "S" majuscule et s'occupe de l'insertion du titre au sein d'une commande `\bprogram`, (Il suffit d'utiliser la fonction chercher/remplacer de votre éditeur \LaTeX sur le mot `section`, mais de façon incrémentale pour ne pas risquer de l'appliquer à un titre suivi de texte en unicolonne)

N.B. l'extension `paracol` est essentielle au projet et aucun extension équivalente ne convient vraiment.

Version 0.6

Des arrangements ont été effectués dans `cpge\lvt.cls` pour apporter (du moins ne plus empêcher) le support du moteur Lua \TeX . L'extension `paracol` est maintenant utilisée pour l'environnement multicolonne. Un mécanisme de marquage (numérotation) des paragraphes a été ajouté. Il est désactivé par défaut.

II Installation et normalisation

II.A Installation des fichiers du projet

Pour rester conforme à l'architecture prévue par le projet il faut :

- prévoir un dossier de travail qui peut porter un nom arbitraire mais qui doit contenir au moins

trois dossiers :

- Base** pour héberger les fichiers de base des programmes ;
- Generic** pour contenir les fichiers maîtres génériques qui permettent de compiler les fichiers de base ;
- Masters** pour contenir les fichiers maîtres de production des livrets.

Voir la section C, [Conventions de nomenclature des fichiers](#) pour plus de détails.

- Installer les fichiers style du projet en suivant la procédure décrite ci-dessus.

Un dossier nommé **texmf** est disponible dans la racine de l'archive du projet. Copier ce dossier et placez-le dans

- votre dossier personnel si vous êtes sous MS Windows ou Linux.

Sous Windows vous pouvez accéder à votre dossier personnel en allant dans `C : \Users \VOTRENOMDUTILISATEUR`, remplacez Users par Utilisateurs si vous utilisez une version française de Windows. Sous Linux, c'est simplement le dossier qui apparaît sous le nom Home dans votre gestionnaire de fichiers.

- le dossier **Library** de votre dossier personnel si vous êtes sous MacOS.

Si vous utilisez une version francisée de MacOS alors ce dossier apparaît sous le nom **Bibliothèque** votre *dossier personnel* dans le Finder. Normalement il est caché donc vous devez suivre la procédure standard pour y accéder.

La suite dépend de votre système d'exploitation et de la distribution **TEX** que vous utilisez.

MacOS il n'y a rien d'autre à faire ;

Linux cela dépend de la distribution Linux. Les fichiers peuvent être directement accessibles sans rien faire d'autre mais parfois vous devez exécuter une ligne de commande dans le Terminal :

```
cd %pour être sur d'être dans votre dossier personnel
mktexlsr texmf
```

MS Windows si vous utilisez MikTeX ouvrez « MikTeX Console » à partir du menu Démarrer et suivez la procédure décrite dans la dernière section nommée « Your own TEXMF root directories » sur la page web suivante (cliquez sur le lien) :

<https://miktex.org/kb/texmf-roots>.

Le dossier **texmf** que vous avez précédemment placé dans votre dossier personnel est compatible TDS, allez donc directement au point 5 et ajouter-le.

Si vous utilisez TexLive la procédure est la même que sous Linux. Ouvrez une console et exécutez la ligne de commande

```
mktexlsr texmf
```

Une fois cette procédure achevée vous pouvez compiler vos fichiers n'importe où sans vous soucier des fichiers style. Vous pouvez même ouvrir le présent manuel avec la ligne de commande usuelle `texdoc lvrtdkit`

Avertissement !

Les fichiers style du projet ont besoin d'une version relativement récente de la distribution \LaTeX utilisée. Que ce soit MikTeX ou TeXLive, une distribution datant d'au moins 2019 est suffisante mais la dernière version disponible est toujours préférable.

Les distributions Linux ont tendance à proposer par défaut de vieilles versions de TeXLive. Pour savoir quelle version de \LaTeX vous avez, c'est simple. Exécutez `pdflatex` en ligne de commande et regardez les premières lignes de ce qui est affiché. Tapez ensuite les touches `ctrl+D` pour interrompre `pdflatex`.

Le cas échéant, vous pouvez soit mettre à jour votre distribution Linux soit récupérer une version complète de TeXLive sur son site officiel et l'installer manuellement. Allez sur cette page par exemple pour télécharger l'image `.iso` de la version que vous voulez :

<https://texlive.info/historic/systems/texlive/>

La procédure d'installation est simple et elle est la même pour tous les systèmes d'exploitation. Toutefois la méthode pour l'intégrer ensuite dans le chemin de recherche des fichiers exécutables dépend de la version Linux que vous utilisez. Les ressources sur Internet qui montrent comment faire ne manquent pas.

Pour MacOS, la meilleure option d'installation consiste en la distribution MacTeX. Elle inclut une version complète de TeXLive plus différents utilitaires (l'éditeur TeXShop par exemple). Plusieurs versions peuvent coexister sur le système sans problème. La dernière installée devient la version par défaut.

II.B Fichiers style du projet

Outre les fichiers de base et les fichiers maîtres décrits dans la section suivante, le projet utilise les fichiers suivants :

- `cpgelvrt.cls`** fichier de classe du projet. C'est le seul fichier dont l'utilisation est visible sur un fichier maître. Le chargement des fichiers qui suivent dans cette description se fait à travers des options de celui-ci. Il contient toute la logique d'organisation des fichiers du projet. Il est capable de générer automatiquement les titres du livret, des parties et des chapitres ainsi que la production des couvertures en utilisant les options de la classe et des commandes principales qu'il définit.
- `lvrtprint.sty`** fichier style qui contient la géométrie de page ainsi que l'espacement des titres et le style entêtes/pieds de page pour les versions imprimables. Il est chargé par défaut ;
- `lvrtscreen.sty`** la même chose pour les versions pour consultation sur écran. Il est chargé en lieu et place du fichier précédent si l'option de classe `screen` est activée ;
- `lvrtcover.sty`** fichier style dont la seule fonction est la production des couvertures. Son activation se fait par le biais de l'option `cover` de la classe `cpgelvrt`.

lvrtcommon.sty fichier style qui contient les définitions des outils utilisables dans les fichiers de base (multicolonne, listes,...) ainsi que la description des styles des titres. C'est le fichier de classe `cpgelvrt.cls` qui s'occupe de son chargement ;

lvrtbase.sty nouveau fichier qui met en place, entre autre, l'interface de configuration clé/valeur et les mécanismes de modification des propriétés des compteurs.

lvrtmath.sty nouveau fichier qui définit des fonctionnalités avancées pour le traitement du texte mathématique ;

Logos dossier contenant les logos au format PDF. Le logo du ministère est utilisé dans la page de garde de chaque document produit et dans les pages de couvertures générées avec l'option `cover`.

En fait `cpgelvrt.cls`, une fois la partie logique initialisée, charge l'un des fichiers `lvrtprint.sty`, `lvrtscreen.sty` ou `lvrtcover.sty` selon les options fournies à la classe.

Où mettre ces fichiers ?

La section précédente expose la procédure à suivre pour centraliser ces fichiers.

Tous les autres fichiers cités ci-après n'ont plus d'utilité. Le choix des polices de caractères disponibles dans une distribution LaTeX est beaucoup plus riche de nos jours et un jeu de polices qui ne nécessite aucune installation supplémentaire est utilisé.

Fonts.zip archive ZIP contenant les fichiers de polices pour une utilisation avec $\text{Xe}_{\text{L}}\text{TeX}$ ou $\text{Lua}_{\text{L}}\text{TeX}$. Il faut désarchiver le fichier de telle façon à obtenir un dossier Fonts contenant les fichiers de polices de caractères. Ce dossier doit obligatoirement rester dans la racine du dossier du projet (à coté des dossiers Base et Masters). Les fichiers de polices présents dans le dossier Fonts ne sont nécessaires que si l'option `fonts` du fichier de classe est activée et que le compilateur utilisé est $\text{Xe}_{\text{L}}\text{TeX}$ ou $\text{Lua}_{\text{L}}\text{TeX}$. Ce sont les polices de production finale des livrets.

Pour diminuer le nombre de fichiers dont dépend le projet, les fichiers suivants ne sont pas disponibles dans l'archive du projet. Ils sont générés automatiquement à la première compilation d'un fichier maître.

gtdnames.def supprimé du projet.

gtdfont.sty fichier style annexe. Il contient la partie relative au choix de polices de caractères et la définition de différentes macros utilisée pour régler les styles de texte dans différentes partie du document. Son utilisation est facultative et ne se fait que lorsque l'option `fonts` de `cpgelvrt.cls` est activée.

Les polices utilisées sont LINUX LIBERTINE, LINUX BIOLINUM et PT SANS. Elles sont normalement disponibles à travers les paquets $\text{L}_{\text{A}}\text{TeX}$ `libertine`, `ptserif` et `ptsans`. En cas d'absence de ces derniers dans votre système, installez les en s'aidant du gestionnaire de "package" de votre distribution $\text{L}_{\text{A}}\text{TeX}$.

gtdxfont.sty son rôle est le même que le fichier précédent, mais dans le cas où le compilateur $\text{Xe}_{\text{L}}\text{TeX}$ (commande `xelatex`) ou $\text{Lua}_{\text{L}}\text{TeX}$ (commande `lualatex`) est utilisé. $\text{Xe}_{\text{L}}\text{TeX}$ et $\text{Lua}_{\text{L}}\text{TeX}$

permettent d'utiliser toutes les polices OPENType installées dans le système d'exploitation. Pour pouvoir utiliser `gtdxfont.sty`, il faut désarchiver le fichier `Fonts.zip` et placer le dossier `Fonts` obtenu dans la racine du dossier du projet.

La création et l'utilisation des fichiers `gtdfont.sty` et `gtdxfonts.sty` ne se fait que si l'option `fonts` du fichier de classe est activée. Dans le cas contraire un choix de polices générique est effectué.

II.C Conventions de nomenclature des fichiers

Pour son bon fonctionnement, le projet exige que les fichiers soient déposés dans des dossiers bien précis selon leur nature. Les rédacteurs de programmes auront à traiter deux fichiers dans la phase de pré-production :

un fichier maître générique qui se contentera d'inclure un fichier de base. L'intervention sur ce fichier est réduite à son minimum. Ces fichiers devraient être créés dans le dossier **Generic** à la racine du dossier du projet.

un fichier de base qui doit contenir le texte du programme en lui même sans la commande `\documentclass` et sans les usuels `\begin{document}` et `\end{document}`. Un fichier par matière/filière/niveau. Des commandes sont mises à leurs disposition pour leurs faciliter la tâche. Ces fichiers devraient être créés dans le dossier **Base** à la racine du projet.

Chaque programme exige donc de travailler avec au moins deux fichiers. La saisie du texte du programme se fait dans le fichier de base placé dans le dossier **Base** et la compilation se fait sur le fichier maître générique correspondant qui est lui placé dans le dossier **Generic**.

Les noms des fichiers de bases doivent être formatés de la façon suivante

`<code matière>` - `<code niveau>` - `<code classe>` . `tex`

Ceux des fichiers maîtres génériques doivent simplement ajouter un préfixe constant à ceux des noms des fichiers de base correspondant sous la forme

`00-prg` - `<code matière>` - `<code niveau>` - `<code classe>` . `tex`

`00-prg` est un préfixe commun à tous les fichiers maîtres génériques.

`<code matière>` code par matière en trois caractères :

mat, phy , chi , sci , inf , gee , gem , eed, egm, egh, ara , fra , ang

`<code niveau>` est l'un des deux codes 1e ou 2e, pour première année et deuxième année;

`<code classe>` est un code en 5 caractères de la classe concernée :

mpsi-, mp---, pcsi-, psi--, pc---, tsi--, ect--, ecs--

II.C.1. Exemples

de nom fichier maître générique : `00-prg-phy-1e-mpsi-.tex`

du nom du fichier de base associé : `phy-1e-mpsi-.tex`

Les fichiers maîtres de production sont à la charge de l'équipe de production. Leur utilisation est traitée dans la dernière section de ce manuel.

II.D Contenu des fichiers

Les fichiers de base ne devraient contenir aucune information active (visible sur le fichier compilé) autre que le texte du programme. Ils ne devraient pas contenir les instructions `\documentclass`, `\begin{document}` et `\end{document}` mais aussi ne pas utiliser la commande `\chapter` car le projet utilise celle-ci dans un but bien précis. Le codage de leurs noms servira à les identifier. Il faudra y utiliser les commandes mise à disposition par les fichiers styles du projet.

Les fichiers maîtres *génériques* ne devront faire l'objet d'aucun ajout autre que l'instruction d'inclusion d'un fichier de base. Toute information supplémentaire sera irrémédiablement perdue dans le processus de production des livrets (les fichiers maîtres génériques ne sont là que pour faciliter le travail des rédacteurs). En exemple, voici ce que pourrait être le contenu du fichier maître `00-prg-chi-2e-mp---.tex`.

```
\documentclass{cpgelvrt}
\begin{document}
%%Informations pour la page de garde et le titre
\setinfo{filiere=mp,niveau=2,matiere=chi}
%% Insérer le titre
\maketitle
\includebase{chi-2e-mp---} % instruction d'inclusion du fichier de base
\end{document}
```

code 1

L'utilité des différentes commandes est facilement compréhensible à partir des lignes commentées du code. Mais précisons que c'est la commande `\includebase` qui s'occupe d'inclure le fichier de base. On lui fournit en argument le nom du fichier à inclure sans aucune indication sur son emplacement car celui-ci est encodé en dur dans les fichiers styles. C'est pour cette raison qu'il est indispensable de respecter l'arborescence du projet.

II.E Normalisation des fichiers de bases

Un fichier de base devrait se limiter au contenu (logique) du programme concerné sans aucun attribut de mise en forme. Un jeu de commandes est prévu par les fichiers styles pour faciliter la saisie et ne pas trop encombrer le code source afin de ne pas nuire à sa lisibilité. Les rédacteurs doivent se conformer autant que possible aux règles suivantes :

- parcourir *complètement ce manuel au moins une fois* pour prendre connaissance des fonctionnalités qu'offrent les fichiers styles. Cela évitera la peine d'ajouter des fonctions ou des effets de style qui sont déjà assurés par les fichiers du projet. Y revenir ou à sa version courte aussi fréquemment que nécessaire.
- depuis un bout de temps déjà, l'encodage des caractères par défaut sous \LaTeX est l'unicode (utf8). Les fichiers styles du projet activent celui-ci. Il n'est pas nécessaire d'utiliser des commandes spéciales pour les lettres accentuées, il suffit de les saisir telles quelles. Cela a en plus l'avantage de ne pas entraver le correcteur orthographique de l'éditeur de texte utilisé.
- les utilisateurs doivent en général veiller à la qualité des codes sources. Ceux-ci doivent rester lisibles et rendre compte de l'organisation logique du contenu ;
- les titres des parties d'un programme doivent être insérés en utilisant les commandes LaTeX pour les titres (`\section`, `\subsection`,...) (voir la section suivante pour plus de détails) ;
- utiliser les outils du projet pour la mise en forme des objectifs (voir la section suivante) ;
- le texte ne doit contenir aucune commande de changement de gras ou de taille des polices (`\textbf`, `\bfseries`, `\large`, `\Large`, ...). Tout ce qui peut tenir pour un titre doit utiliser les commandes de sectionnement \LaTeX . Pour mettre en valeur un passage de texte utiliser la commande LaTeX à connotation plus sémantique `\emph` ;
- les listes doivent être écrites en utilisant les environnements \LaTeX à cet effet (`itemize`, `enumerate` et `description`) ou les outils spécifique au projet (`\pcit`) et non pas de simples symboles d'énumération et des retours forcés à la ligne ;
- toute commande explicite produisant des espaces verticaux (`\vspace`, `\vskip`...) est à proscrire. Ces commandes peuvent s'avérer très nuisibles et le projet prévoit des solutions de remplacement moins brutales pour la plupart des situations ;
- il faut éviter d'utiliser des commandes de rupture de page « brutales » comme `\newpage` ou `\eject`. Penser au fait que les mêmes fichiers sources servent à produire des documents de plusieurs formats. Des outils plus souples sont mis en place par les fichiers style ;
- l'utilisation des tableaux LaTeX n'est pas désirable mais ils peuvent s'avérer nécessaires. Afin d'optimiser les espaces verticaux et les ruptures de page dans le produit final, les tableaux doivent soit être flottants s'il ne dépassent pas une page (en les encadrant dans un environnement `table`), soit utiliser l'environnement `tprogram` prévu par le projet. L'environnement `tprogram` est capable, contrairement à l'environnement standard `tabular`, de s'étendre sur plusieurs pages.

II.F À propos de typographie

La typographie est universelle. Ses règles dépendent des traditions dans la langue utilisée et du support de destination. Certaines sont franches et incisives, d'autres sont plus permissives ou plus nuancées. Pour le projet il importe de suivre les règles suivantes :

- ne pas utiliser le souligné pour mettre en valeur du texte. C'est un héritage des machines à taper où il n'y avait pas d'autres moyens pour différencier typographiquement un texte.

- il n'y pas de convention fixe en ce qui concerne le **gras**. Pour le projet il est réservé aux titres, aux numéros des items dans les listes et en général pour souligner une hiérarchie dans le contenu.
- pour mettre en valeur un passage dans le texte utiliser de l'italique si le texte alentour est droit et inversement.

LaTeX

C'est exactement ce que fait la commande `\emph`.

- dans le document compilé, les symboles de ponctuation qui ne surmontent pas la ligne de base du texte (point, virgule et points de suspension) doivent être accolés au mot précédant et suivis d'une espace. Ceux qui le font (; : ? ! « ») doivent être précédés et suivis d'une espace.

LaTeX

L'extension `babel` chargée avec l'option `french` par les fichiers style gère automatiquement les espaces autour d'une ponctuation. Elle rend les caractères de ponctuation actifs (ils agissent comme des commandes) donc pas besoin de les faire précéder d'un espace, mais l'espace suivant doit être présent. Lorsque c'est nécessaire l'espace typographique qu'ils insèrent avant est dite insecable dans le sens qu'une rupture de ligne est rendue impossible à son niveau. Traditionnellement on utilise aussi la commande `~` mais ce n'est pas nécessaire si on ne laisse pas d'espace entre la ponctuation et le mot précédent.

Ces règles ne sont pas appliquées dans le mode mathématique donc il faudra ajouter les espaces nécessaires manuellement. En outre une formule centrée est traitée comme du texte normal, elle peut donc finir par une ponctuation selon sa position dans la phrase.

Dans le texte normal, les points de suspension (...) s'insèrent avec simplement trois points successifs et non avec la commande `\cdots` ou similaire car celles-ci sont destinées au mode mathématique qui a ses propres conventions typographiques.

- Pour les différents types de tirets, y compris leur utilisation sous LaTeX, consulter la page Wikipedia (cliquer sur le lien) :

<https://fr.wikipedia.org/wiki/Tiret>

- La typographie des tableaux est très exigeante. Il n'est par exemple pas conseillé d'utiliser des filets verticaux pour séparer les colonnes et surtout pas des filets trop épais car ils attirent de façon inconsciente le regard au lieu de servir de guide visuel. Garder en tête que dans un tableau, c'est le texte qui constitue l'information principale et tout effet de style doit être fait dans le but de faciliter sa lecture.

LaTeX

Le package `booktabs` est chargé par défaut par les fichiers style du projet. Il propose une alternative aux filets par défaut LaTeX et son manuel explique pourquoi certaines habitudes sont contraires à la typographie des tableaux.

Il stipule d'utiliser les commandes `\toprule` et `\bottomrule` pour les lignes d'ouverture et de fermeture du tableau et, sans en abuser, la commande `\midrule` au milieu, voir `\cmidrule` pour les lignes avec cellules fusionnées. Il déconseille avec insistance l'usage des filets verticaux. Tous

les tableaux de ce manuel l'utilisent et respectent ces consignes.

II.G Rigide ou élastique

Les sauts entre les mots sur une ligne et ceux entre les lignes et les paragraphes sur une page ne sont pas de dimensions rigides. \TeX va les allonger ou les rétrécir selon le besoin pour conserver un bon équilibre des vides sur la page. C'est l'essence même de ce que fait \TeX et ce qui le distingue des autres solutions de traitement de texte. En fait \TeX prévoit deux méthodes pour créer des espaces, ceux qui sont rigides et ceux qui sont élastiques. Les documentations mal faites omettent de mentionner cet aspect ce qui mène à des habitudes incorrectes. Par ailleurs les commandes qui permettent de gérer les dimensions dans LaTeX (`\newlength`, `\setlength`...) ne font (juste en apparence) pas de distinction entre les deux types ce qui amène encore plus de confusion. Par exemple, la plupart des documentations ne mentionnent pas assez clairement *qu'il ne faut jamais* insérer manuellement des sauts verticaux rigides, sauf dans des cas très particuliers. Cela gêne les algorithmes de rupture de page du moteur \TeX et peut rendre le travail de post-production intenable.

Selon le jargon \TeX , une dimension rigide est de type `dimen` et une dimension élastique et de type `skip`. La première se présente sous forme d'un nombre flottant suivi d'une unité. La seconde sous forme d'un nombre flottant muni d'une unité avec optionnellement des valeurs plus et/ou minus sous la forme :

6pt plus 1pt minus 0.5pt

plus et minus sont des mots clés qui doivent être écrits explicitement tels quel. La première mesure est la valeur absolue du `skip`, les deux autres sont ses composantes d'élasticité. Un saut qui doit utiliser un `skip` va utiliser la valeur absolue de la dimension et au besoin ajouter au plus la valeur plus ou retrancher au plus la valeur minus. Les dimensions rigides sont utilisées pour des espaces qui doivent toujours être les mêmes, comme par exemple les marges de la page, la taille d'une boîte, l'épaisseur d'un filet, l'espace de séparation entre le texte et le cadre dans `\fbox`... Les dimensions élastiques sont utilisés pour effectuer des sauts. C'est la clé qui garantit des lignes et des pages avec des espaces vides équilibrés.

\TeX permet de déclarer une nouvelle variable `dimen` avec la commande `\newdimen`, et une nouvelle variable `skip` avec la commande `\newskip`. La commande \LaTeX `\newlength` crée en fait une variable de type `skip`. On peut contrôler la valeur d'une variable de dimension avec la primitive `\showthe`. Si vous mettez quelque part une instruction de la forme `\showthe\baselineskip` ou `\showthe\oddsidemargin` la compilation \TeX s'arrête à ce niveau et montre les valeurs contenues dans ces variables. Ils suffit de taper sur la touche « Enter » pour que la compilation continue. Si on utilise une dimension rigide là où \TeX attend un `skip`, il l'absorbera comme valeur absolue du saut sans s'en plaindre mais ne pourra pas augmenter ou diminuer le saut. Si on utilise une dimension élastique là où \TeX attend une `dimen` il va simplement utiliser la valeur absolue de la dimension et écrire tout logiquement ce qui reste dans le document final.

Ce n'est pas que \LaTeX est déficient de ce côté. Tout au contraire, son but ultime est de décharger

l'utilisateur de ces détails techniques pour le laisser se concentrer sur l'organisation sémantique du contenu qu'il rédige. Par exemple, il a bien prévu des commandes pour effectuer des sauts verticaux correctement avec des dimensions plus ou moins grandes selon le besoin : `\smallskip`, `\medskip` et `\bigskip`. Ces commandes effectuent des sauts verticaux qui, comme on peut le deviner, se font avec des `skip`. Ces dimensions sont respectivement enregistrées dans les variables `\smallskipamount`, `\medskipamount` et `\bigskipamount`. On peut contrôler les valeurs qu'elles contiennent avec `\showthe` et si besoin les altérer avec `\setlength`. Mais, svp n'allez pas titiller des variables comme `\baselineskip` au risque de complètement bousiller l'harmonie de votre document. Pour fermer cette parenthèse les sauts horizontaux, et encore plus les sauts verticaux, doivent être effectués avec des `skip`. Pour le projet, il n'est souvent pas nécessaire d'utiliser *explicitement* des commandes de saut.

Des mécanismes plus sûrs

Les fichiers style du projet mettent à disposition des mécanismes pour altérer les sauts verticaux de façon plus rapide et moins risquée.

Fin de paragraphe ou retour à la ligne

Quand on laisse une ligne vide on informe T_EX que le paragraphe précédent est fini. C'est la même chose que d'insérer explicitement la commande `\par`. Dans cette situation T_EX fait de ce point un très bon candidat pour une rupture de page. Bien plus qu'un point quelconque au milieu du paragraphe. La commande `\\` a une signification particulière dans certains environnements comme `tabular` ou `align`. On n'en parle pas ici. Donc quand la commande `\\`, ou son équivalent `\newline` est utilisée en dehors de ces environnements elle force le passage à la ligne mais sans TERMINER LE PARAGRAPHE courant. Elles découragent même encore plus une rupture de page à leur niveau. Utilisez maintenant systématiquement `\\` pour revenir à la ligne et T_EX sera obligé d'aller chercher des ruptures de page ailleurs. Cela peut aboutir à des anomalies plus ou moins sévères.

On ne s'en rend pas compte sur de petits documents, mais avec des documents longs les ruptures de page peuvent devenir un vrai casse-tête surtout quand on utilise des éléments flottants, des tableaux, plusieurs colonnes... comme dans le projet. Autant éviter de gêner T_EX dans ce qu'il sait très bien faire par lui-même. N'utilisez `\\` que lorsque vous avez besoin d'un alignement spécifique du texte sans rompre l'intégrité logique du paragraphe (comme pour les vers d'un poème). Autrement laissez une ligne vide.

Penalty !

Qui a parlé de pénalités ? C'est un autre aspect que les documentations évitent de discuter, à tort ou à raison. Y compris celle-ci.

`\baselineskip` et `\parskip`

`\baselineskip` est un `skip` qui sert comme saut entre deux lignes dans un même paragraphe. Il est calculé dynamiquement à chaque changement de taille de police avec les commandes `\large`, `\Large`... ou doit être fourni explicitement avec la commande plus générale `\fontsize`. À chaque changement de taille, il faut que la commande `\par` soit à la portée de la commande qui provoque le changement, dans le même groupe T_EX. C'est ainsi qu'une instruction de la forme

code 2

```
... du texte ... \\
{\large Un text en plus grande taille} \\
... encore du texte
```

donne un interligne incorrect après le texte en `\large`. Celui associé à la taille de police en dehors du bloc `{ }`. La façon correcte de faire la même chose est plutôt

code 3

```
... du texte ...

{\large Un text en plus grande taille \par}
... encore du texte
```

Le skip `\parskip` est utilisé comme saut qui s'ajoute à `\baselineskip` pour séparer deux paragraphes. Dans les réglages par défaut de \LaTeX il est nul car on préfère marquer les paragraphes avec une indentation au début. D'autres conventions optent plutôt pour une indentation nulle et un saut plus grand pour distinguer les paragraphes. La première convention convient à des textes avec des paragraphes longs. Elle donne un moins bon résultat pour les textes avec essentiellement des paragraphes courts. C'est le cas des livrets et c'est donc cette dernière convention qui est adoptée.

II.H Plaidoyer pour les solutions de multicolonnage

Le projet a besoin d'une fonctionnalité bien précise : permettre de saisir le texte du programme officiel sur une colonne et de manière synchronisée les commentaires sur une autre.

Une solution pourrait être d'utiliser des tableaux à deux colonnes. Toutefois cette piste apporte plus de problèmes qu'elle n'en résout. Pour ne lister que les plus gênants :

- bien que des packages permettant d'étaler des tableaux sur plusieurs pages existent depuis longtemps, il est impossible de faire éclater le contenu d'une cellule sur deux pages. Une cellule qui survient en fin de page doit alors être éclatée en plusieurs cellules, parfois en contradiction avec l'intégrité logique du texte, tout en risquant de tout devoir refaire en cas de changement du contenu.
- le code source est difficilement lisible dans un tableau à cause des instructions de formatage qui peuvent parfois être très lourdes pour obtenir certains effets (comme pour la fusion de cellule par exemple). Cela rend le maintien des fichiers sources et leurs évolution difficile.
- la gestion des erreurs est très mauvaise sur les tableaux. En gros le compilateur renvoie à la ligne de fin du tableau. Ce qui risque d'être gênant si on traite un tableau qui est sensé occuper plusieurs pages. On peut y remédier en découpant le contenu sur plusieurs tableaux de plus petite taille, mais cela ne résout pas les autres problèmes.
- La synchronisation source/PDF ne fonctionne pas avec les packages pour grands tableaux.

La synchronisation source/PDF est une fonctionnalité indispensable. Elle rend possible d'aller directement au point correspondant dans le fichier source à partir d'une position dans le document PDF. On peut difficilement s'en passer quand on veut apporter des modifications à un fichier PDF qui contient une centaine de page.

Le package `paracol` pour contenu synchronisé en multicolonne offre des environnements \LaTeX qui répondent exactement aux besoins du projet tout en n'ayant aucun de ces défauts. Le projet offre une surcouche logique de ce dernier qui facilite grandement son utilisation. Pour résumer trois commandes permettent d'obtenir l'effet voulu. Ce qui a l'avantage de donner des fichiers sources très lisibles.

Les rédacteurs de contenu sont ainsi encouragés à se familiariser avec ces outils et à éviter autant que possible l'utilisation des tableaux.

III Traitement des fichiers de base

III.A Conventions de notation

Cette section est éminemment technique. Toutes les commandes décrites répondent à des besoins réels qui n'ont pas tous été prévus dès l'élaboration du projet mais qui se sont dégagés, explicitement ou implicitement, des habitudes des différents intervenants en ce qui concerne la composition de texte avec le système \LaTeX . Elles ont été pensées pour offrir une bonne ergonomie, être facile à saisir, refléter le sens sémantique du texte qu'elle traitent tout en maintenant une bonne lisibilité du code source.

Les commandes suivent des conventions communes respectant globalement les standards \LaTeX avec quelques innovations propre au projet. Les paramètres qu'elles utilisent sont de trois types :

- les paramètres *obligatoires*. Ils doivent être fournis entre `{ }`. Une commande peut en avoir plusieurs ou aucun et ils sont toujours, à quelques exceptions près, placés en dernier ;
- les paramètres *optionnels* entre `[]`. Quand un tel paramètre est présent, en général c'est un texte qui va être repris tel quel quelque part dans le document final avec éventuellement certains effets, comme par exemple dans `\item[<text>]` ou `\section[<short title>]{<title>}`. Certaines commandes du projet peuvent en avoir plusieurs contrairement aux habitudes LaTeX (au plus un paramètre optionnel en général) ;
- les paramètres *optionnels* entre `< >`. Ce sont tous des listes d'options de type `<clé>=<valeur>` séparés par des virgules qui servent à personnaliser le comportement de la commande. Quand un tel paramètre est disponible pour une commande, il est toujours unique et, à une exception près, il doit être placé en premier, immédiatement après le nom de la commande sans aucun espace.
- L'effet du changement que les options entre `< >` induisent est en général local et ne concerne que le texte à la portée de la commande. Toutefois il y a des options qu'on peut régler globalement

pour tout le document. Il faudra alors les fournir à une commande de configuration spéciale (`\lvrtsetup`) et non seulement en option à une commande donnée. On n'utilise plus pour cela les caractères `< >` pour les encadrer mais `{ }`.

La description d'une commande commence dans la marge et est formée du nom de la commande ainsi que de la spécification précise de tous les paramètres qu'elle gère. La nature de chaque paramètre est donnée explicitement entre les deux symboles `< >` :

| Ne pas confondre `< >` et `< >`.

- `<text>` un texte normal qui va en général être repris tel quel dans le document ;
- `<int>` un nombre entier positif ;
- `<num>` une valeur numérique flottante, souvent comprise entre 0 et 1 pour indiquer une proportion. Utilisé exclusivement pour les dimensions rigides.
- `<frac>` une fraction qui doit être insérée par exemple sous la forme $3/2$ pour indiquer une proportion. Toutes les commandes et les options qui impliquent des sauts verticaux utilisent ce type de paramètre ;
- `<dim>` une dimension LaTeX rigide, une épaisseur ou dans certains cas des « espaces » rigides (qui a dit sauts rigides) ;
- `<skip>` une dimension LaTeX élastique, en général un saut vertical ;
- `<color>` le nom d'une couleur qui doit avoir été prédéfinie ;
- `<keyval>` une liste de couples de type `<clé>=<valeur>` qui servent en général à modifier le comportement d'une commande.
- `<char>` utilisé comme valeur dans certaines options `<clé>=<valeur>`. Il s'agit d'un caractère unique qui va déclencher un certain comportement de la commande ;
- `<bool>` utilisé dans les options `<clé>=<valeur>`. C'est un commutateur logique. À l'utilisation on doit soit écrire simplement son nom sans la partie `=<val>` auquel cas il devient vrai, soit écrire `<bool>=false` auquel cas il devient faux. Comme par exemple `landscape` et `landscape=false` quand on utilise le package `geometry` ;
- `<code>` une instruction LaTeX qui sera utilisée dans le corps de la commande ;
- `<cmdname>` certaines commandes du projet permettent de construire d'autres commandes pour un usage particulier comme le font par exemple `\newcommand` elle-même ou `\newtheorem`. On dira que ce sont des méta-commandes. Ce type de paramètre doit commencer avec `\` et indique le nom de la commande qui sera créée par ces constructeurs ;
- `<csname>` un texte normal sans nombre et sans caractères spéciaux et qui ne commence pas avec `\` qui sera utilisé dans la formation du nom d'une commande qui sera créée par une méta-commande.
- `<list>` une liste d'éléments qui sont de même nature séparés par des virgules. En général utilisé comme valeur dans certaines options sous la forme `<opt>={<list>}`. Il faudra alors veiller à l'encadrer entre accolade pour masquer la virgule à l'intérieur car celle-ci est aussi utilisée pour séparer les options principales.

- * c'est un caractère optionnel qui est utilisé en général pour modifier le comportement de la commande.

À chaque fois que c'est nécessaire une liste de descriptions des options $\langle \text{clé} \rangle = \langle \text{valeur} \rangle$ est donné avec une indication de leurs valeurs par défaut si elles en ont qui apparaît à droite sur la même ligne. Si l'option admet un raccourci (un nom plus court) alors celui-ci s'affiche à gauche de la description juste en bas de son intitulé.

Pourquoi $\langle \text{num} \rangle$ et $\langle \text{frac} \rangle$

L'arithmétique selon T_EX est contraignante. Lorsqu'on multiplie un skip par un $\langle \text{num} \rangle$ alors il est coercisé en une dimen. Pour préserver son élasticité il faut le multiplier par une $\langle \text{frac} \rangle$.

Jargon

Dans l'univers L^AT_EX on parle parfois de version étoilée d'une commande $\backslash \text{cmd}$ comme si $\backslash \text{cmd}^*$ était une commande à part entière alors qu'en fait dans cette écriture $*$ ne fait pas partie du nom de la commande. La commande $\backslash \text{cmd}$ teste simplement si elle est suivie du caractère $*$ et modifie son comportement dans ce cas. C'est différent pour les environnements. Dans $\backslash \text{begin}\{\text{align}^*\}$, par exemple, $*$ fait bien partie du nom de l'environnement.

L^AT_EX en marche

Au moins une commande du projet utilise un autre caractère comme modificateur. Il s'agit de la « commande » $\backslash \text{pcit-}$. Le principe est le même qu'avec $*$. Les nouvelles versions de L^AT_EX (depuis au moins 2019) permettent d'utiliser facilement n'importe quel « token » comme modificateur alors que les anciennes versions n'offraient aucune méthode utilisateur officielle et utilisaient en interne une technique manuelle fastidieuse et limitée au caractère $*$. Regarder la documentation du package [xparse](#) ou chercher le document [usrguide.pdf](#) dans le dossier d'installation de votre distribution (texdoc est votre ami). Les nouveautés ne se limitent pas aux dits modificateurs. Une vraie richesse a été ajoutée à la manière de définir de nouvelles commandes.

Bookmarks PDF

Pour le présent manuel, chaque description de commande, d'environnement ou d'option insère un lien dans les bookmarks PDF pour permettre de facilement de la retrouver.

Signalons pour clore cette partie qu'il y a deux familles de commandes de configuration : celles qui commencent avec $\backslash \text{lvtset}$ et celles qui commencent avec $\backslash \text{lvtsetup}$. Les secondes ne prennent qu'un seul argument obligatoire qui doit être une liste de $\langle \text{clé} \rangle = \langle \text{valeur} \rangle$. Les premières sont hautement spécialisées et utilisent chacune sa propre syntaxe selon le besoin sans recourir à un système clé/valeur.

III.B Titres de sections et table des matières

III.B.1. Sections et table des matières générales

Les commandes LaTeX usuelles pour insérer des titres sont disponibles pour une utilisation dans les fichiers de base à l'exception de la commande `\chapter`. Il est fortement recommandé de les utiliser chaque fois qu'il y a besoin de mettre du texte en gras ou dans une taille plus grande (tout ce qui peut tenir pour un titre). Leurs utilisation doit respecter le découpage suivant :

`\section*``[<text>]{<text>}`

Commande LaTeX conventionnelle pour insérer le titre d'une section. Pour rappel, comme pour les autres commandes dans cette section :

- le caractère `*` est utilisé en option pour signifier que le titre ne doit pas être numéroté et qu'aucune entrée ne doit être ajoutée à la table des matières.
- le paramètre optionnel `[<text>]` est utilisé quand le titre est trop long et qu'on désire qu'une alternative plus courte apparaisse dans la table des matières tout en conservant le titre original `{<text>}` dans le texte.

Pour le projet, Cette commande est utilisée pour les grandes parties du programme, thermodynamique par exemple pour la Physique. En absence de `*`, cette commande produit un numéro et crée une entrée dans la table des matières ;

`\subsection*``[<text>]{<text>}`

à utiliser pour les chapitres du programme. En absence de `*`, cette commande produit un numéro et crée une entrée dans la table des matières ;

`\subsubsection*``[<text>]{<text>}`

`\paragraph*``[<text>]{<text>}`

`\subparagraph*``[<text>]{<text>}`

à utiliser s'il y a besoin de plus de subdivisions logiques dans un chapitre du programme. Ses commandes sont réglées pour ne pas produire de numéros ni créer des entrées dans la table des matières.

`\setcounter``{secnumdepth}{<int>}`

C'est l'instruction standard LaTeX qui permet de régler la profondeur au delà de laquelle les numéros n'apparaissent plus avec les titres. Par défaut les fichiers styles lui donnent la valeur 3, ce qui limite la numérotation à la commande `\subsection`. Si un rédacteur tient vraiment à faire augmenter cette profondeur pour que, disons, `\subsubsection` insère aussi un numéro, il suffit d'utiliser au début du fichier de base l'instruction

`\setcounter{secnumdepth}{4}`

Pourquoi pas `\chapter`

La commande `\chapter` est utilisée en interne par les fichiers styles du projet. Elle est insérée automatiquement en début de chaque fichier de base quand on l'inclut dans le fichier maître avec la commande `\includebase`. Le titre affiché peut changer selon le type de livret à produire. Il est « calculé » à partir des informations fournies dans le fichier maître.

Table des matières

La table des matières est ainsi limitée aux noms des parties et ceux des chapitres des programmes. Si besoin, sans rien changer au contenu des fichiers de bases, il est possible d'y faire apparaître des entrées pour plus de titres mais contrairement à la numérotation des titres cette tâche revient à l'équipe de production finale. Les réglages s'effectueront dans les fichiers maîtres et non dans les fichiers de base.

III.B.2. La commande `\subchapter`

Dans certaines matières il y a besoin d'une subdivision supérieure à celle des grandes parties. Par exemple en physique et en chimie où il y a une partie « Formations expérimentale » et une partie « Contenu thématique » pour les cours théoriques ou dans d'autres matières où il y'a une « Première période », « Deuxième période »... Pour combler cette lacune, une unité de subdivision intermédiaire entre `\chapter` et `\section` a été ajoutée. Il s'agit de la commande `\subchapter`.

`\subchapter*` [`<text>`] {`<text>`}

assure toutes les fonctionnalités attendues d'une commande de titrage (référence croisées, insertion dans la table des matières, utilisation dans les entêtes/pieds de page). Sa syntaxe est exactement la même que les autres commandes.

III.B.3. Titres des sessions de TP

Un traitement spécial est accordé aux titres des sessions de TP.

`\tpsection` <`int`> [`<text>`] {`<text>`}

`\settpsection`

`\unsettpsection`

La commande `\tpsection` est propre au projet. Elle permet d'insérer le titre d'une unité de TP. Elle fonctionne comme les commandes de titrage normale \LaTeX à part qu'elle n'a pas de version étoilée et qu'elle possède un paramètre optionnel en plus entre `< >`. Elle utilise aussi sa propre numérotation et celle-ci n'est pas résiliée à chaque changement de `\section`.

<`int`> l'entier `<int>` est utilisé pour indiquer le nombre de séance que doit durer le TP. Si ce paramètre est utilisé, cette information apparaît dans le titre et dans la table des matières avec des formatages spécifique qui les met en valeur. Le texte « `<int>` séance(s) » est ajouté à gauche du titre et « (`<int>`s) » apparaît avec le numéro de page dans la table des matières.

Il est possible de changer localement pour un fichier de base le terme séance et son abréviation s avec la commande `\seancename`. Par exemple :

```
\seancename{heure}{h}
```

Une telle instruction devrait figurer au début du fichier de base pour qu'elle soit facilement accessible.

[`<text>`] et {`<text>`} ont les fonctions habituelles pour une commande de titrage.

Si la commande `\tpsection` est utilisé sans aucune précaution elle provoque une erreur comme quoi elle n'est pas définie. Elle a besoin de réglages spécifiques et il faut préparer d'abords le contexte. Pour le faire utiliser la commande `\settpsection` juste avant de commencer à lister les titres des TP. Une fois ceci terminé il faudra tout remettre dans son état original avec la commande `\unsettpsection`.

Un exemple (extrait d'un fichier de base pour le programme de chimie) :

```
\section{Solutions aqueuses}
\settpsection
\tpsection{Initiation aux TP de chimie. Préparation de solutions aqueuses.}
\begin{itemize}
\item Sélectionner et utiliser le matériel adapté à la précision requise.
\item Distinguer les instruments de verrerie In et Ex.
\item Préparer une solution de concentration en masse ou en quantité de
matière donnée à partir d'un solide, d'un liquide, d'une solution de composition
connue avec le matériel approprié.
\end{itemize}
\tpsection[Dosages pH-métrique et conductimétrique acide fort/base forte]{Dosages
pH-métrique et conductimétrique acide fort/base forte (choix d'un indicateur de fin
de réaction).}
. . . . .
\unsettpsection
```

code 4

`\tpsection` fait appel dans sa définition à la commande `\subsection`. Ce qui permet de faire apparaître la liste des TP dans la table des matières. Le formatage spécifique du titre et de la table des matières est mis en place avec `\settpsection` et il est annulé avec `\unsettpsection`.

III.B.4. Styles des titres

On a vu comment configurer le niveau jusqu'auquel les titres affichent un numéro avec la méthode \LaTeX officielle. En outre, les fichiers styles mettent en ouvre un mécanisme propre au projet pour personnaliser ces numérotations et éventuellement le style global des titres. Ces mécanismes peuvent être utilisés dans un fichier de base à condition d'avoir une raison pertinente de le faire. Il va de soit que le style des titres doit rester homogène dans tout le livret.

Les deux commandes `\lvtsetcounter` et `\lvtsetstyle` utilisent un mécanisme de mot clés pour changer rapidement le contenu de la numérotation (son style). Un mot clé est un caractère unique qui doit être inséré entre deux barres verticales `|`. C'est une représentation simple du style qui doit être utilisé pour un numéro selon le [tableau III.1](#). Le style par défaut pour `\section` et `\subsection` est par exemple obtenu avec les instructions

```
\lvtsetstyle{section}{|I|} % chiffre romains en majuscule
\lvtsetstyle{subsection}{|S|.1|} % numéro de section et chiffre arabe
```

code 5

code	style	visuel
1	chiffres arabes	1,2,3...
n	chiffres arabes avec symbol n°	n° 1, n° 2, n° 3...
a	lettres en minuscule	a,b,c...
A	lettres en majuscule	A,B,C...
i	chiffres romains en miniscule	i,ii,iii...
I	chiffre romain en majuscule	I,II,III...
m	numérotation ordinale au masculin	premier, deuxième...
f	numérotation ordinale au féminin	première, deuxième...
S	numéro de \section en cours	
s	numéro de \subsection en cours	

TABLE III.1 – Raccourcis utilisables dans les numérotations des titres

On peut par exemple activer l'apparition des numéros dans \subsubsection et changer le style de numérotation avec

```
\setcounter{secnumdepth}{4}
\lvertsetstyle{subsubsection}{|s|. |a|}
```

code 6

Le numéro contiendra le numéro de la \subsection en cours et le numéro de \subsubsection en lettres minuscules. La commande \lvertsetstyle a donc pour fonction de changer le style d'un numéro. La commande \lvertsetcounter est plus générale, elle permet de configurer plus de propriétés du compteur, y compris son style. Sa syntaxe est de la forme

\lvertsetcounter{<nom>}{<style>}[<parent>](<char>)

Change les propriétés du compteur <nom>. Les paramètres utilisables sont :

- <nom> paramètre obligatoire qui doit être le nom du compteur objet de la modification. Si ce compteur n'existe pas, il est créé.
- <style> paramètre obligatoire qui fixe le style qui lui sera appliqué selon les conventions décrites ci-dessus ;
- <parent> est un paramètre optionnel qui peut être une liste de noms de compteurs existants optionnellement précédés d'un signe -. Sans le signe moins, le compteur objet de la modification sera remis à zéro chaque fois que le compteur de la liste change. C'est ce qui arrive par exemple pour subsection chaque fois qu'on utilise \section. Avec le signe - c'est le contraire de l'opération qui se produit : le compteur à modifier devient indépendant du compteur de la liste. Si on veut par exemple que le compteur de subsection ne change plus avec \section, mais continuer à dépendre de \chapter il suffit de placer au début

```
\lvertsetcounter{subsection}{|1|}[-section,chapter]
```

code 7

⟨char⟩ c'est un paramètre optionnel (entre parenthèses). Il doit être un caractère unique qui sera ensuite utilisé comme raccourci pour la valeur en cour du compteur ⟨nom⟩, à la manière de |S| pour section.

Le compteur subsection n'a par exemple pas de raccourci. Si on veut en créer un pour l'utiliser dans \paragraph et faire afficher les numéros pour ceux si, il suffit donc de mettre en début du fichier de base

```
\setcounter{secnumdepth}{5}
\lvertsetstyle{subsubsection}{|s|. |a|}(|k|)
\lvertsetstyle{paragraph}{|k|. |1|}
```

code 8

Maintenant une situation invraisemblable juste pour illustration : si on veut que paragraph ne soit pas résilié à chaque \subsection mais que la numérotation soit en continu pour tout le fichier de base tout en affichant le numéro courant de subsection, il suffit d'insérer les instructions

```
\setcounter{secnumdepth}{5}
\lvertsetstyle{subsubsection}{|s|. |a|}(|k|)
\lvertsetcounter{paragraph}{|k|. |1|}[-subsubsection, -subsection, -section]
```

code 9

\lvertsetdeco{⟨nom⟩}{⟨code⟩}

Sert à modifier la décoration avec laquelle sera affiché le compteur ⟨nom⟩. Le code ⟨code⟩ doit être un code L^AT_EX valide qui peut utiliser #1 pour représenter la valeur courante du compteur avec son style exactement comme si on utilisait la commande \newcommand pour définir la décoration. Un exemple d'utilisation pourrait être

```
\lvertsetdeco{subsubsection}{\llap{\bfseries #1.\enskip}}
```

code 10

La commande \llap sert à insérer le numéro sur la marge et \enskip ajoute un petit espace qui est à peu près celui occupé par le caractère n. Le rôle de \lvertsetdeco est d'éviter de surcharger le style de numérotation avec des éléments décoratifs car c'est celui-ci qui apparaît lorsqu'on fait une référence avec \ref sur le titre.

Les trois commandes \lvertsetcounter, \lvertsetstyle et \lvertsetdeco ne concernent pas seulement les styles des numérotations des titres. Elles sont utilisées pour changer ce style pour d'autres compteurs. Ceux utilisés par enumerate par exemple.

Les commandes suivantes par contre sont exclusivement dédiées aux commandes de titrage.

\lvertsetfont{⟨nom⟩}{⟨code⟩}

Sert à imposer le style avec lequel sera affiché le titre associé au compteur ⟨nom⟩. ⟨code⟩ doit être

un code \LaTeX valide sans arguments qui sera placé avant le titre et son numéro. Peut être utilisé pour changer la taille ou la graisse du titre par exemple.

`\lvertsetspacing{<nom>}{<dim>}{<skipb>}{<skipa>}`

Sert à régler les espaces utilisés par le titre associé au compteur `<nom>`. `<dim>` est l'espace d'indentation du titre par rapport au texte normal. Il peut être négatif. Les dimensions élastiques `<skipb>` et `<skipa>` sont utilisées comme sauts avant et après le titre. Au lieu de fournir la spécification complète (en tant que `skip`) de ces deux paramètres on peut utiliser des valeurs de la forme `*<num>` où `<num>` est un nombre qui sera utilisé comme facteur multiplicatif par une valeur étalon prédéfinie. Par exemple :

```
\lvertsetspacing{subsubsection}{\opt}{*2}{*1}
```

code 11

pour une indentation nulle, un saut 2 fois plus grand que la normale avant et un saut normal après le titre.

Et encore !

Voir la section [Des titres moins contraignants](#) pour la commande `\customtitle` qui permet de créer des titres moins contraignants et qui s'intègrent mieux avec l'environnement multicolonne.

III.B.5. Tables des matières partielles

Encore à finaliser...

III.C Objectifs et ordre

Deux environnements sont prévus pour la mise en forme des objectifs : `objectif` et sa version étoilée `objectif*`.

`\begin{objectif*}`
`\end{objectif*}`

à utiliser avec des textes assez longs. Ceux d'introduction à la matière par exemple. Il ne produit pas de décoration autour du texte mais s'arrange pour que l'apparence soit bien différente du texte du programme.

`\begin{objectif}`
`\end{objectif}`

à utiliser pour les textes courts. Ceux d'introduction à une partie du programme par exemple.

`\begin{ordre}`
`\end{ordre}`

Dans le cas où le préambule contient une consigne sur l'ordre des chapitres à suivre, utiliser l'environnement de liste `ordre`. Il propose un format distingué pour accentuer l'importance de son contenu. C'est un environnement qui fonctionne comme `enumerate`, les unités pouvant être insérées avec `\item`.

III.D Programme et commentaires

Pour les matières scientifiques, à part les textes d'introduction, des objectifs et diverses précisions, le contenu du programme est organisé en deux colonnes. La colonne de droite pour le contenu officiel du programme, synchronisée avec une colonne à gauche pour les commentaires.

III.D.1. Les commandes principales

Trois commandes suffisent pour traiter le contenu en multicolonne : `\bprogram`, `\bcomment` et `\eprogram`.

`\bprogram`

commande pour insérer un paragraphe du programme (volet de gauche). Une commande `\bprogram` peut être suivie d'une autre commande `\bprogram` si le paragraphe ne possède pas de commentaire. Cette commande se charge aussi d'ouvrir l'environnement multicolonne quand il ne l'est pas.

`\bcomment`

commande pour insérer un commentaire à droite d'un paragraphe `\bprogram`. Le commentaire sera aligné verticalement avec le paragraphe du programme qui l'a précédé. Une commande `\bcomment` peut être suivie d'une autre commande `\bcomment` si le commentaire suivant n'est associé à aucun paragraphe du programme. Si l'environnement multicolonne n'a pas été ouvert (avec `\bprogram`) alors `\bcomment` s'en charge.

`\bprogram` et `\bcomment` ne prennent pas d'argument donc inutile d'encadrer le texte suivant entre accolades. Leurs effet se résume en fait au déplacement dans la colonne de destination et l'activation du style spécifique au texte de la colonne (police de caractère, alignement...). Le contenu est aussi isolé (dans un « groupe » \TeX) donc tout changement d'état (`\bfseries`, `\itshape`...) qui y survient est annulé au changement de colonne suivant.

`\eprogram`

Ferme l'environnement multicolonne.

Normalement, il est faut fermer l'environnement multicolonne pour insérer du texte en pleine largeur (un titre ou les objectifs par exemple). Mais dans certaines situations c'est inutile de le faire. Un mécanisme est disponible pour insérer du contenu en pleine largeur sans quitter l'environnement bicolonne.

Pour résumer

une fois l'environnement multicolonne initialisé avec le premier `\bprogram` (ou le premier `\bcomment`), on peut changer de côté avec les mêmes commandes autant que nécessaire. On ne ferme l'environnement multicolonne (avec `\eprogram`) qu'une fois qu'on veut insérer de grands blocs de texte en pleine largeur. Une rupture de page peut survenir au milieu d'un même bloc programme/commentaire sans gêner la synchronisation verticale des deux colonnes. La saisie est simple et le code reste très clair puisqu'un bloc peut contenir autant de texte qu'il faut sans avoir à l'encadrer entre des accolades (ni d'un environnement spécifique). Les erreurs pointent vers les endroits exacts où elles surviennent et la synchronisation PDF/source reste fonctionnelle.

Avertissement

Un environnement multicolonne ne doit pas être ouvert au milieu d’une liste. Que ce soit les listes `TeX itemize`, `enumerate`, `description` et plus généralement `list`, mais aussi celles propres au projet, `objectif`, `objectifs*`, et toutes les listes créées avec `\pcit` ou `\pcnum`^a. La conséquence serait un déséquilibre du texte sur les deux colonnes.

a. voir la section F

Pour insérer du texte en pleine largeur sans quitter l’environnement bicolonne il suffit de le fournir en argument optionnel à la commande `\bprogram`. Par exemple

```
\bprogram[\subsubsection{<un titre>}]
< ... paragraphe du programme ... >
```

code 12

Le titre est insérée en pleine largeur et le texte qui suit la commande est inséré dans la colonne de gauche de façon normale. La seule limitation est que le texte passé en argument ne doit pas contenir lui-même les caractères `[]`, par exemple ceux encadrant une option d’une commande. Dans ce cas il suffit d’encadrer l’argument en entier entre deux accolades pour masquer les caractères `[]` à l’intérieur.

Selon la même idée on peut ajouter par exemple un filet horizontal (pour mieux imiter un tableau) sur toute la largeur du texte avec la primitive TeX `\hrule` avec quelque chose comme

```
\bprogram[\vspace{6pt}\hrule\vspace{3pt}]
```

code 13

Cette instruction a des défauts. Une commande spéciale est prévue pour cette tâche.

Maintenant comment altérer la largeur des deux colonnes ou en général régler différents aspects de l’environnement. La dernière version du projet prévoit un mécanisme clé/valeur pour le faire. C’est à la première commande `\bprogram` qui initialise l’environnement multicolonne (mais aussi à `\bcomment` si c’est elle qui initialise l’environnement) qu’incombe cette tâche. La syntaxe complète de ces deux commandes est de la forme :

```
\bprogram<⟨keyval⟩>[⟨code⟩]
\bcomment<⟨keyval⟩>[⟨code⟩]
```

[⟨code⟩] est comme on l’a vu n’importe quelle instruction LaTeX qui sera exécutée sur la pleine largeur de la page.

<⟨keyval⟩> et une liste de couples **<clé>=⟨valeur>** fournie entre **<>** qui permet de régler certains aspects de l’environnement. *Si cet argument est utilisé alors que l’environnement est déjà initialisé alors il est ignoré.* Les options utilisables sont comme il suit.

ratio=⟨num⟩ déf : 0.5

⟨num⟩ un nombre flottant entre 0 et 1 qui précise le ratio que doit occuper la colonne à gauche sur la largeur de la ligne courante. Celui de la colonne à droite s’en déduit automatiquement. La valeur

par défaut est 0.5.

head= $\langle text \rangle \{ \langle text \rangle \}$ déf : {Programme}{Commentaires}

active l'insertion d'un entête pour l'environnement à la manière d'un tableau. Les deux arguments (obligatoires) fixent le contenu à gauche et à droite de l'entête. Un filet de terminaison est aussi inséré à la cloture de l'environnement avec `\eprogram`. Une utilisation systématique de cette option n'est pas recommandée. Elle est plus utile pour des cas particuliers en dehors du couple `program`/`commentaires`.

sep width= $\langle dim \rangle$ déf : 20pt

$\langle dim \rangle$ est une dimension LaTeX qui précise l'espace de séparation entre les deux colonnes.

seprule width= $\langle dim \rangle$ déf : 0pt

$\langle dim \rangle$ est l'épaisseur du filet de séparation des deux colonnes. Par défaut il est nul.

seprule color= $\langle color \rangle$ déf : 

$\langle color \rangle$ est le nom d'une couleur qui sera utilisée pour le filet de séparation des colonnes.

seprule= $\{ \langle dim \rangle \} \{ \langle color \rangle \}$

pour fixer à la fois l'épaisseur et la couleur du filet de séparation. C'est la même chose que d'utiliser :
`seprule width= $\langle dim \rangle$, seprule color= $\langle color \rangle$`

before= $\langle code \rangle$

$\langle code \rangle$ est un code \LaTeX arbitraire qui sera exécuté juste avant l'ouverture de l'environnement multicolonne.

after= $\langle code \rangle$

$\langle code \rangle$ est un code LaTeX arbitraire qui sera exécuté juste après la fermeture de l'environnement multicolonne.

before skip= $\langle skip \rangle$ déf : `\parskip`

$\langle skip \rangle$ est une dimension élastique qui sera utilisée pour effectuer un saut vertical juste avant l'ouverture de l'environnement multicolonne. Par défaut c'est le saut de paragraphe normal.

interskip factor= $\langle frac \rangle$ déf : 2/3

une fraction qui servira dans le calcul du saut vertical entre deux rangées du mode multicolonne. Le saut est égal à `frac` multiplié par le saut de paragraphe du mode normal (celui en vigueur en dehors du mode multicolonne). Le saut de paragraphe au sein d'une même cellule est lui toujours nul.

interskip= $\langle skip \rangle$

permet de directement fixer le saut inter-rangées sans passer par un calcul intermédiaire. Il va de soit qu'il faut bien renseigner les composantes d'élasticité dans la valeur donnée.

after skip= $\langle skip \rangle$ déf : `\parskip`

$\langle skip \rangle$ est une dimension élastique qui sera utilisée pour effectuer un saut vertical juste après la fermeture de l'environnement multicolonne.

left style= $\langle code \rangle$

$\langle code \rangle$ est un code LaTeX qui fixera le style utilisé dans la colonne du texte du programme.

right style=*<code>*

<code> est un code LaTeX qui fixera le style utilisé dans la colonne des commentaires.

Par exemple pour fixer une largeur à droite plus petite et au même temps utiliser de l'italique pour les commentaires

```
\bprogram<ratio=0.6, right style=\itshape>
< ... texte programme ...>
\bcomment
< ... texte commentaire ...>
.....
\eprogram
```

code 14

Comme signalé auparavant, ses options sont ignorées si l'environnement est déjà en cours.

Si on a déjà utilisé ces options et qu'on veut les réutiliser plutard pour un autre environnement, ce n'est pas la peine de les resaisir, il suffit de mettre simplement

```
\bprogram<restore>
```

code 15

pour les réactiver pour l'environnement en cours d'initialisation. L'option **restore** restaure *le dernier lot d'options utilisées*.

\lvertsetuppc{*<keyval>*}

Tout ce mécanisme a un effet local. Les options ne sont appliquées qu'à l'environnement en cours d'initialisation. Pour les fixer globalement, il faudra utiliser la commande **\lvertsetuppc** avec les mêmes options décrites ci-dessus en dehors de tout environnement \LaTeX . L'effet sera appliqué aux environnements qui seront ouverts après. Par exemple

```
\lvertsetuppc{ratio=.6, comment style=\itshape, beforeskip=\medskipamount,
afterskip=\medskipamount}
```

code 16

fera que tout environnement bicolonne ouvert après l'occurrence de cette instruction adoptera le style de l'exemple précédent plus l'ajout d'un saut vertical avant et après chaque environnement multicolonne. Saut qui sera le même que celui ajouté par la commande **\medskip**.

Pour finir, il est possible au début du fichier de base de donner un nom à un lot d'options et ensuite de fournir ce nom comme option à la commande **\bprogram** pour l'appliquer. la syntaxe est de la forme :

\lvertsetuppc{save = {*<nom>*}{*<keyval>*}}

<nom> sera utilisé comme « nom » des options *<keyval>*. Pour utiliser ces options ensuite il suffit d'initialiser l'environnement avec **\bprogram<nom>**. En exemple :

```
\lvrtsetuppc{save={1tier}{ratio=.33}}
\lvrtsetuppc{save={2tier}{ratio=.66}}
```

code 17

créera deux options nommées `1tier` et `2tier` qui fixent respectivement la largeur de la colonne à gauche au tiers et au 2 tiers de la largeur de la zone de texte moins l'espace de séparation des colonnes. Ces noms pourront ensuite être utilisés seules ou avec d'autres options dans le paramètre `< >` de `\bprogram` comme dans

```
\bprogram<2tier, righth style=\itshape>
```

code 18

`\bprogram<restore>` utilise un mécanisme similaire. Chaque fois que le paramètre `< >` est utilisé avec `\bprogram`, les options sont enregistrées avec un nom interne et sont restaurées si l'option `<restore>` est utilisée. À chaque nouvelle utilisation de l'argument `< >`, ces options sont mises à jour (sauf bien sûr si on se limite à `<restore>`).

III.D.2. Outils annexes

Une fois l'environnement bicolonne initialisé, on peut y utiliser les commandes décrites ci-après.

`\interrule<keyval>`

insère un filet de séparation horizontal en pleine largeur et se déplace vers la première colonne. Les options utilisables sont : `b = <dim>` espace avant ; `a = <dim>` espace après ; `h = <dim>` épaisseur du filet ; `c = <color>` couleur du filet ;

`\interskip{<frac>}`

insère un saut vertical en pleine largeur qui est égal à `<frac>` multiplié par le saut de paragraphe externe et reste sur la colonne courante ;

Ces deux commandes produisent des erreurs si elles sont utilisées en dehors du mode multicolonne.

`\pcsec \pcsubsec \pcsubsubsec \pcpara \pcsubpara`

sont des versions des commandes de titrage L^AT_EX qui insèrent les titres suggérés par leurs noms sur toute la largeur de la zone de texte alors qu'on est au milieu d'un environnement multicolonne sans avoir à le fermer. Si elles sont utilisées en dehors de l'environnement alors elles agissent comme les commandes normales.

III.D.3. Marquage des paragraphes du programme

Un mécanisme est prévu pour la numérotation des paragraphes des programmes. Cela pourrait s'avérer utile lorsque deux parties différentes veulent communiquer à propos d'un bout du programme.

Deux stratégies sont adoptées :

`\markparacom`

\unmarkparacom

la première est globale et permet de numéroté chaque paragraphe du programme inséré après une commande `\bprogram`. Le compteur utilisé est réinitialisé à chaque utilisation de la commande `\section`. Pour utiliser cette option il suffit de placer quelque part dans le fichier de base la commande `\markparacom`. Le changement prend effet après la ligne où elle a été insérée. Pour annuler son action, il suffit de placer la commande `\unmarkparacom`.

\nbprogram

l'autre permet un ajustement plus fin de la numérotation des paragraphes. Il suffit d'utiliser `\nbprogram` en lieu et place de `\bprogram` pour placer un numéro à gauche du paragraphe qui suit la commande. De cette façon on arrive à créer une division en des unités logiques du programme. Il n'est pas nécessaire avec cette méthode d'utiliser les commutateurs `\markparacom` et `\unmarkparacom`.

\renewcommand\theparacom{<description>}

permet de redéfinir le style du numéro de paragraphe. En fait, il s'agit d'un compteur (paracom) et la commande présentée ici est celle usuelle pour le changement du style d'un compteur \LaTeX . Le style par défaut est fixé par :

```
\renewcommand\theparacom{%
  \footnotesize(\arabic{section}.\arabic{paracom})}
```

code 19

Il est possible de désactiver le marquage des paragraphes sur un fichier quelque soit la méthode utilisée pour l'activer. Il suffit de placer dans le fichier maître qui l'appelle la commande `\disablemarks`. La commande `\reenablemarks` permet de réactiver le marquage si ce dernier a été désactivé par `\disablemarks`.

Exemple de rendu avec la commande `\nbprogram` :

- | | |
|--|---|
| <ol style="list-style-type: none"> 1 Qu'arrive-t-il si on insère une note de bas de page ¹ ou un objet flottant au milieu d'un environnement multicolonne ? 2 Qu'arrive-t-il si on insère une note de bas de page ou un objet flottant ² au milieu d'un environnement multicolonne ? | <p>Les <i>footnotes</i> d'une colonne ne sont plus insérées en bas de la même colonne. Les objets flottants sont gérés de façon plus complexe.</p> <p>Les <i>footnotes</i> sont gérées par colonnes : les footnotes ³ d'une colonne seront insérées de façon normale en bas de page.</p> |
|--|---|

III.E Des titres moins contraignants

Regardons maintenant comment insérer des titres plus résilients dans le sens où ils sont indépendants de l'architecture globale des titres sous \LaTeX et se comportent convenablement en mode multicolonne

-
1. commande `\footnote`
 2. avec les environnements `figure` ou `table`
 3. et les objets flottants aussi

en ce qui concerne les sauts qui les encadrent. On a parfois besoin d'insérer un intitulé, au milieu d'un tableau ou du mode multicolonne, sans que cela ne déclenche toute la machinerie sous-jacente des titres \LaTeX mais sans concessions au niveau typographique et stylistique. Un titre ne s'improvise pas en plaçant un numéro et du texte en gras ou en plus grande taille. Le minimum est de répondre aux exigences suivantes :

- les *sauts* implicites avant et après le titre doivent respecter les conventions ;
- de préférence l'intitulé doit être indenté par rapport à un éventuel numéro ;
- les numéros *doivent* être générés dynamiquement pour ne pas avoir à tout refaire en cas d'ajout ou de suppression d'une unité ;
- un titre doit bien se comporter s'il se trouve à proximité de la fin de page pour ne pas avoir à forcer une rupture de page pour éviter un intitulé orphelin.

Le projet dispose désormais d'une commande qui répond, au moins aussi bien que les commandes \LaTeX , à ces exigences et qui permet de créer de nouveaux styles de titres sans toutefois aller jusqu'à l'intégration complète dans les mécanismes \LaTeX (pas de gestion de la table des matières par exemple). Elle est par contre assez polyvalente dans le sens où :

- elle détecte si elle est utilisée dans un environnement multicolonne et adopte un comportement plus adéquat ;
- elle prépare le contexte pour une éventuelle utilisation des références croisées, y compris sur l'intitulé lui même (avec la commande usuelle `\nameref`) ;
- on peut facilement créer une hiérarchie entre titres qui est indépendante de celle des titres standards.

`\customtitle*`[*<text>*]{*<text>*}

S'il est présent, l'argument dans [*<text>*] est utilisé comme numéro, sinon un compteur interne est utilisé pour la numérotation. L'argument dans {*<text>*} sert comme texte du titre. Avec le modificateur `*` le titre est inséré dans la colonne courante, sans lui il l'est sur la pleine largeur de la page. Une commande `\label` insérée après le titre crée correctement une ancre qui peut être référencée ailleurs avec `\ref` ou avec `\nameref`. On notera en particulier que la signification des paramètres est très différente de celle des commandes de titrage usuelles (qui sont hors-propos ici, puisque la gestion de la table des matières n'est pas disponible). Un numéro est toujours produit sauf si on utilise un argument optionnel vide [].

Il est en outre possible d'utiliser un compteur personnalisé, éventuellement créé pour l'occasion :

```
\lvertsetcounter{theme}{|1} % au début du fichier de base
\customtitle<c=theme>{Le titre}
```

code 20

Le compteur `theme` est incrémenté et ensuite inséré comme numéro. Ceci peut être utile pour créer

des commandes de titrage au comportement correct à moindre effort. Il suffit par exemple de mettre au début du fichier de base

```
\lvertsetcounter{theme}{|1|}[section]
\newcommand\Theme{\customtitle<c=theme>} % c pour counter
```

code 21

pour disposer ensuite de la commande `\Theme` qui agit presque comme `\customtitle` mais avec son propre compteur. Dans l'exemple précédent celui-ci est résilié à chaque utilisation de `\section`.

Il est aussi possible de régler, par exemple, la taille à laquelle le numéro et le titre s'affichent :

```
\lvertsetcounter{theme}{|1|}[section]
\newcommand\Theme{\customtitle<c=theme, s=\large>} % s pour style
```

code 22

La commande ainsi créée ne gère toutefois plus le paramètre `<>` qui sert à altérer localement son comportement. Pour créer un clone de `\customtitle` associé à un compteur et des options prédéfinis, il faut utiliser la méta-commande `\newsimplifiedtitle` :

```
\newsimplifiedtitle\Theme{s=\large, bs=3/2, as=1}
% bs pour before skip
% as pour after skip
```

code 23

Plus besoin de spécifier un compteur car un compteur interne est automatiquement créé pour la commande. Les options données en deuxième argument fixent le comportement par défaut de la commande mais peuvent ensuite être écrasées avec le paramètre `< >` comme pour `\customtitle`. Les options utilisées ici précisent les sauts avant et après le titre. Elles sont exprimées comme fractions de sauts étalons définis par les fichiers style. En outre le nom interne du compteur créé est `cTheme`. Ses propriétés peuvent donc être réglées avec les commandes `\lvrsetcounter` ou `\lvtsetstyle`, voir `\lvtsetdeco`.

La syntaxe générale de cette méta-commande est de la forme :

`\newsimplifiedtitle<cmdname>{<keyval>}`

Crée une commande de nom `<cmdname>` (qui doit commencer avec `\`) qui se comporte comme `\customtitle` à la différence qu'elle adopte par défaut les options fournies dans `<keyval>`. Un compteur est automatiquement créé et associé à la commande. Son nom est celui du nom de la commande sans `\` et précédé du caractère `c`. Les propriétés de ce compteur peuvent ensuite être ajustées avec `\lvtsetcounter` et consœurs.

Les options utilisables sont avec `\customtitle` et `\newsimplifiedtitle` sont :

use counter=`<text>`

rac : **c**

`<text>` est le nom d'un compteur qui doit avoir été créé auparavant avec la commande `\lvtsetcounter`.

La commande standard `\newcounter` n'est pas suffisante car les compteurs qu'elle crée ne gèrent pas

les mot-clés. Le compteur est utilisé juste pour le titre en cours. Il n'est pas associé définitivement à la commande, à moins de définir une nouvelle commande comme dans

```
\newcommand\Theme{\customtitle<c=theme>}
```

Cette option n'a pas d'intérêt dans `\simplifiedtitle` car un compteur est construit à partir du nom de la commande fourni en premier argument.

title style= $\langle code \rangle$ déf : opt

rac : s $\langle code \rangle$ est un code \LaTeX qui est incrusté juste avant le texte titre et son numéro. Son effet est limité au titre. Il permet, entre autre, de fixer la taille à utiliser.

title indent= $\langle dim \rangle$

rac : i dimension qui est utilisée pour indenter le titre par rapport au texte normal. Elle peut être négative.

labelsep= $\langle dim \rangle$ déf : 0.5em

rac : ls espace de séparation entre l'intitulé et le numéro.

before skip= $\langle frac \rangle$ déf : 1

rac : bs fraction qui sera utilisée dans le calcul du saut vertical avant le titre par rapport à une valeur étalon. Sa valeur par défaut est 1 en dehors du mode multicolonne. Dans celui-ci ce saut est annulé et le titre se contente du saut entre rangée.

after skip= $\langle frac \rangle$ déf : 1

rac : as fraction qui sera utilisée dans le calcul du saut vertical après le titre par rapport à une valeur étalon.

break factor= $\langle frac \rangle$ déf : 3

rac : bf une dimension spécifique égale à cette fraction multipliée par le saut de ligne normal est créée. S'il reste moins d'espace dans la page que cette dimension alors une rupture de page est déclenchée avant le titre. Sert à éviter les titres orphelins en fin de page. Normalement un effort est fait pour que cela ne se produise pas en des circonstances normales. Mais parfois \TeX se trouve à court d'options, comme dans le cas où deux titres se suivent ou que le titre est suivi d'un élément trop haut comme un tableau par exemple. Voir la section sur les ruptures de page pour plus de détails.

runin= $\langle bool \rangle$ déf : true

rac : r variable booléenne. Si elle est activée alors le titre sera inséré en mode runin, c'est à dire que le texte suivant est placé en continuité du titre sur la même ligne. C'est par exemple le comportement par défaut de la commande `\paragraph` dans \LaTeX .

Un exemple complet maintenant : dans certaines matières, on a besoin d'une subdivision bien précise à utiliser dans l'environnement bicolonne. Il y a des thèmes et chacun est décomposé en plusieurs unités. On insère donc au début du fichier de base de la matière le code suivant

```
\newsimplifiedtitle\theme{s=\large}
\newsimplifiedtitle\unite{afterskip=2/3}
\lvertsetcounter{ctheme}{|I|}[section](T)
\lvertsetdeco{ctheme}{Thème #1.}
```

code 24

```
\lvertsetcounter{cunite}{|T|.1|}[theme]
```

Ce qui crée les commandes `\theme` et `\unite`. Le compteur `ctheme` qui s’affiche avec le style « **Thème I., Thème II....** » est remis à zéro après chaque `\section`. Un raccourci est créé pour `ctheme` et il est utilisé dans le style de `cunite`. Celui-ci s’affiche donc avec le style « **I.1., I.2....** » et il est automatiquement remis à zéro après chaque commande `\theme`. Une hiérarchie peut ainsi être mise en place entre les titres.

III.F Des listes plus expressives

Un environnement de liste (`itemize`, `enumerate`, ...) insère un espace vertical supplémentaire à son début et à sa fin. Ceci posait problème dans les versions précédentes quand un tel environnement commence le texte sur une colonne. La liste ne commençait plus à la même hauteur que le texte qui lui est associé dans l’autre colonne. Ce problème a été réglé dans la version courante. En outre de nouvelles structures ont été ajoutées pour faciliter l’insertion de listes avec une paramétrisation plus poussée que ce que offre LaTeX par défaut et une organisation plus explicite au niveau du code `LaTeX`. Elles sont utilisables dans le mode multicolonne comme dans le mode normal et elles changent de comportement selon le contexte. Dans un environnement multicolonne elles annulent les espaces verticaux et utilisent des espaces horizontaux plus compacts tout en continuant à respecter les conventions typographiques des listes.

Listes... ou munitions

Les listes sont le nerf de la guère de la composition de texte en classes préparatoires. Pas un seul document ne s’en prive. `LaTeX` manque d’une interface de personnalisation simple de leurs propriétés (indentation, style de numérotation...). Des packages existent pour le faire. L’un des plus notables est `enumitem`. Il est utilisé en interne par le projet tout en ajoutant une couche d’abstraction qui simplifie l’accès à ses fonctionnalités. En outre un nouveau paradigme dans l’utilisation des listes est disponible non seulement pour les fichiers du projet mais *dans tout fichier LaTeX qui utilise l’instruction* `\usepackage{lvtcommon}`.

III.F.1. Tutoriel

Il s’avère que les listes à puce (`itemize`) sont utilisées dans les fichiers de base bien plus fréquemment que les listes numérotées. Commençons donc par elles.

`\pcit` `\pcclose`

La commande `\pcit` fonctionne comme `\item` avec des différences notables :

- on n’a pas besoin d’ouvrir un environnement particulier pour l’utiliser. Quand elle est lancée pour la première fois, elle ouvre elle-même un environnement `itemize` et insère un `item`;
- une fois l’environnement ouvert, elle se comporte comme la commande `\item` normale. On peut d’ailleurs utiliser `\item` à la place.

- pour fermer l'environnement `itemize` initialisé par le premier `\pcit`, utiliser la commande `\pcclose`. Si toute la séquence se produit dans un environnement multicolonne, il n'est pas nécessaire d'utiliser `\pcclose`.

Un exemple rapide pourrait être

```
\pcit Un premier item ;
\pcit un deuxième ;
\pcit et un dernier.
\pcclose
```

code 25

Résultat du code 25 :

- Un premier item ;
- un deuxième ;
- et un dernier.

Maintenant comment peut-on imbriquer deux listes : `\pcit` possède un paramètre `< >` dès qu'elle en trouve un, même vide, elle ouvre une nouvelle liste `itemize`. Pour retourner à la liste de niveau supérieure il suffit d'utiliser l'instruction `\pcit-`. Dans ce cas s'il y a encore un niveau ouvert alors un item est inséré dans ce niveau sinon rien ne se passe. Dans ce sens `\pcit-` permet aussi de fermer l'environnement si elle est insérée dans le niveau le plus extérieur.

```
\pcit premier item dans le niveau 1 ;
\pcit deuxième item dans le niveau 1 ;
  \pcit<> premier item dans le niveau
    2 ;
  \pcit second item dans le niveau 2.
\pcit- dernier item dans toute la liste.
\pcclose
```

code 26

Résultat du code 26 :

- premier item dans le niveau 1 ;
- deuxième item dans le niveau 1 ;
 - premier item dans le niveau 2 ;
 - second item dans le niveau 2.
- dernier item dans toute la liste.

Sur cet exemple, remplacer `\pcclose` par `\pcit-` aura le même effet. Maintenant, pour souligner la hiérarchie des items on aimerait bien que la puce de la liste intérieure soit différente. On y arrive en l'indiquant en option :

```
\pcit premier item dans le niveau 1 ;
\pcit deuxième item dans le niveau 1 ;
  \pcit<d> premier item dans le niveau
    2 ;
  \pcit second item dans le niveau 2.
\pcit- dernier item dans toute la liste.
\pcclose
```

code 27

Résultat du code 27 :

- premier item dans le niveau 1 ;
- deuxième item dans le niveau 1 ;
 - ◆ premier item dans le niveau 2 ;
 - ◆ second item dans le niveau 2.
- dernier item dans toute la liste.

L'option `d` qui a servi à changer la puce ici est l'une des options dédiées à cet usage et qui sont résumées dans le tableau

e	s	d	b	t	c
–	■	◆	•	►	✓
<code>\eit/\eclose</code>	<code>\sit/\sclose</code>	<code>\dit/\dclose</code>	<code>\bit/\bclose</code>	<code>\tit/\tclose</code>	<code>\cit/\cclose</code>
endash	square	diamond	bullet	triangle	checkmark

La dernière ligne est fournie à titre indicatif pour aider à mémoriser les initiales qui servent comme options.

Mais au lieu d'utiliser une option pour changer de puce et augmenter le niveau on peut aussi utiliser les commandes de la troisième ligne du tableau précédent. Elles fonctionnent de la même façon que `\pcit/\pcclose` à l'exception que chacune utilise la puce indiquée par la première lettre de son nom et provoque une erreur si on tente de la changer avec une option. À l'usage, pour imbriquer deux listes de cette famille il est inutile d'utiliser `< >` non plus. Un simple changement de nom de commande suffit pour ouvrir ou fermer un niveau.

```
code 28
\sit premier item dans le niveau 1 ;
\sit deuxième item dans le niveau 1 ;
    \dit premier item dans le niveau 2 ;
    \dit second item dans le niveau 2.
\sit dernier item dans toute la liste.
\sit-
```

Résultat du code 28 :

- premier item dans le niveau 1 ;
- deuxième item dans le niveau 1 ;
 - ◆ premier item dans le niveau 2 ;
 - ◆ second item dans le niveau 2.
- dernier item dans toute la liste.

Le dernier `\sit-` ferme le niveau supérieur de la même façon que si on utilise `\bclose` à la place. Constaté que pour créer un nouveau niveau et pour en sortir, aucune indication n'a été fournie aux commandes dans cette construction. On peut même aller plus loin dans les niveaux :

```
code 29
\sit premier item dans le niveau 1 ;
\sit deuxième item dans le niveau 1 ;
    \dit premier item dans le niveau 2 ;
    \dit second item dans le niveau 2.
        \bit un niveau en plus.
\sit dernier item dans toute la liste.
\sit-
```

Résultat du code 29 :

- premier item dans le niveau 1 ;
- deuxième item dans le niveau 1 ;
 - ◆ premier item dans le niveau 2 ;
 - ◆ second item dans le niveau 2.
 - un niveau en plus.
- dernier item dans toute la liste.

Cette fois, l'avant dernière commande `\sit` ne s'est pas contenté de fermer le niveau en cours. elle est revenue au niveau auquel elle a été lancée la première fois.

La saisie en devient beaucoup plus légère et la hiérarchie des items est naturelle et transparente sur le texte \LaTeX . Une solution qui est bien meilleure que de simplement utiliser les raccourcis très populaires

```
code 30
\newcommand\bi{\begin{itemize}}
\newcommand\ei{\end{itemize}}
```

Le fonctionnement global est simple à saisir mais dans le détail l'algorithme utilisé par ces commandes commandes est le suivant :

- si la commande est suivi d'une option `<⟨keyval⟩>` alors elle ouvre une nouvelle liste et insère un item; si c'est une commande spécialisée qui est utilisée pour la première fois elle ouvre un niveau et n'utilise `<⟨keyval⟩>` que pour les options; si un éventuel caractère - est présent après `<⟨keyval⟩>`, il est ignoré (il n'a pas de sens ici);
- sinon :
 - ◆ elle vérifie le niveau auquel elle est exécuté et revient le cas échéant au niveau auquel elle a été lancée; ensuite :
 - si elle est suivie d'un caractère - alors elle recule encore d'un niveau et selon s'il y a encore un niveau ouvert ou pas, place un nouvel item ou ne fait rien.
 - sinon elle insère simplement un item comme le ferait la commande `\item`.

Par ailleurs, toutes les commandes de fermeture sont des synonyme d'une même commande qui porte le nom `\closeslist`. Les noms spécifiques ne sont là que pour appuyer la sémantique du code source. Une commande plus puissante est disponible pour cette tâche : `\shutall` ferme récursivement tous les niveaux déjà ouverts et elle est inoffensive si aucun ne l'est. Cette commande est automatiquement lancée par chaque commande `\bprogram` ou `\bcomment`. Il est donc inutile de fermer une liste dans l'environnement multicolonne si *elle termine le contenu d'une cellule*.

Un autre exemple, cette fois dans un environnement bicolonne qui est lui même personnalisé :

code 31

```

\bprogram<
  ratio=.66,
  sep width=18pt,
  head={Colonne large}{Colonne serrée},
  right style=\sffamily\itshape,
  seprule
>
Texte de programme dans le volet à gauche, parallèle à un commentaire qui commence
par une liste mais avec un alignement vertical correct.
\bcomment
\pcit<b> item 1;
  \pcit<t> sous-item 1;
  \pcit sous-item 2. Pas besoin de fermer les listes, même sur plusieurs niveaux.
\eprogram

```

Résultat du code 31 :

Colonne large	Colonne serrée
Texte de programme dans le volet à gauche, parallèle à un commentaire qui commence par une liste mais avec un alignement vertical correct.	<ul style="list-style-type: none"> • <i>item 1;</i> ▶ <i>sous-item 1;</i> ▶ <i>sous-item 2. Pas besoin de fermer les listes, même sur plusieurs niveaux.</i>

Le paramètre `<keyval>` ne se limite pas à la modification de la puce. On peut y utiliser toutes les options du package `enumitem`. Les curieux se confieront à sa documentation officielle (texdoc `enumitem`) pour en savoir plus. Cela permet de régler très finement les propriétés des listes, y compris pour des utilisations originales comme dans l'exemple suivant :

```
\pcit<labelindent=2cm>[] \small\sffamily
Un paragraphe très indenté par rapport au texte normal. Le paramètre vide $[\;]$ sert
ici à faire disparaître la puce de la liste. Il a le même sens qu'avec la commande
\cmd{item} normale sauf qu'ici l'espace de séparation entre la puce et le texte
suivant est absorbé.\par
L'effet de l'indentation dure même pour plusieurs paragraphes ; tant que
l'environnement n'a pas été fermé. Les listes \LaTeX{} sont parfois effectivement
utilisées de cette manière. C'est ce que font certains environnemnt pour théorèmes
par exemple.
\pcclose
```

code 32

Résultat du code 32 :

Un paragraphe très indenté par rapport au texte normal. Le paramètre vide `[]` sert ici à faire disparaître la puce de la liste. Il a le même sens qu'avec la commande `\item` normale sauf qu'ici l'espace de séparation entre la puce et le texte suivant est absorbé.

L'effet de l'indentation dure même pour plusieurs paragraphes ; tant que l'environnement n'a pas été fermé. Les listes \LaTeX sont parfois effectivement utilisées de cette manière. C'est ce que font certains environnemnt pour théorèmes par exemple.

III.F.2. La syntaxe

La syntaxe complète de la commande `\pcit` et des commandes de la famille `\bit` est de la forme :

```
\pcit<keyval>><->[<text>]
\bit<keyval>><->[<text>]
```

le paramètre optionnel `[<text>]` agit comme avec le `\item` normal : la puce usuelle de la liste est remplacée par `<text>` pour l'item courant. `<keyval>` est une liste d'options, soit propres au projet

comme pour les puces soit des options du package `enumitem`. Le modificateur `-` est utilisé pour fermer le niveau courant ou bien toute la liste si le niveau en cours est le niveau le plus extérieur. Il est absorbé et ignoré si le paramètre `<keyval>` est présent.

De plus, avec les commandes de la famille `\bit`, le paramètre `<keyval>` n'est utilisé que pour passer d'autres options que celle du choix de la puce. Sa présence, même vide, provoque en outre l'ouverture d'un nouveau niveau de liste. Le modificateur `-` n'est pas non plus nécessaire pour fermer le niveau en cours puisque chacune de ces commandes revient à son niveau d'origine en l'absence de `<keyval>`. Dans les deux situations, utiliser une commande au nom différent permet d'augmenter le niveau ou au contraire d'en fermer selon l'algorithme décrit ci-dessus. La commande de base `\pcit` se comporte aussi de la même façon si elle est mélangée avec les autres commandes.

`\newsimplifieditemize{<csname>}{<keyval>}[<char>]`

cela crée les commandes `\<csname>it/\<csname>close` au même comportement que `\pcit/\pcclose` ou bien `\bit/\bclose`. `{<keyval>}` est utilisé pour fixer le comportement par défaut de la commande. Ce comportement peut être ensuite changé localement avec le paramètre `<keyval>` de la commande ainsi créée. Sans le paramètre optionnel `[<char>]`, une commande au comportement similaire à `\pcit` est créée : on peut changer localement de puce. S'il est présent, il fixe la puce qui sera utilisée par la liste et donc crée une commande similaire à `\bit`.

Mais alors quel intérêt ? Puisqu'on dispose de la commande générique `\pcit` et des commandes spécialisées qui épuisent toutes les puces prédéfinies. L'intérêt peut résider dans la définition de nouvelles commandes qui créent des listes avec des propriétés différentes grâce à `enumitem`. La commande générique `\pcit` est par exemple elle-même définie par

`\newsimplifieditemize{pc}{s}`

La seule option déclarée ici est `s` qui fixe la puce par défaut de la commande. L'absence du paramètre `[<char>]` lui permet de changer optionnellement de puce. Ajouter d'autres options avec `s` permet par exemple d'avoir une indentation plus prononcée ou des sauts entre items plus réduits.

On peut aussi définir de nouvelles puces et les associer à des commandes spécialisées. À titre d'indication, le code complet pour créer `\bit` est le suivant

```
\def\lvrtbullet{${\bullet$}
\lvrtlistloadingit{b/.style={puce=\lvrtbullet}}
\SetEnumitemKey{b}{before=\lvrt@enumitemerr{b}}
\newsimplifieditemize{b}{}[b]
```

code 33

La première ligne crée la puce. La deuxième déclare l'option qui sera associée à la puce. La troisième utilise le système de gestion de clés du package `enumitem` pour provoquer une erreur si une commande spécialisée utilise l'option `b` et la dernière définit les commandes `\bit` et `\bclose`.

On peut aussi, de façon plus simple, créer de nouvelles commandes au comportement modifiés à partir des commandes existantes en utilisant le nouveau mécanisme de définition de commandes de

LaTeX. Comme par exemple

```
\NewDocumentCommand\ibibit{d<> o}{%
  \IfValueTF{#1}{\pcit<#1>}{\pcit}
  \IfValueT{#2}{\texttitle{\itshape #2}}}%
}
\let\ibibclose\closelist
```

code 34

Ce qui crée la commande `\ibibitem` qui se comporte à peu près comme `\pcit` :

- si un paramètre `< >` est présent, il est relayé à la commande `\pcit` sous-jacente ;
- le paramètre optionnel `[]` est lui intercepté pour insérer son contenu avec le style utilisé pour les titres et en italique. La puce est par contre toujours produite.
- la commande `\ibibclose` permet de fermer la liste.

`\ibibit` imite de ce fait le style utilisé pour formater une bibliographie. Le paramètre optionnel `[]` devrait être le titre de l'ouvrage. Pas d'association possible avec la commande `\cite` par contre. C'est simplicité mais peut être suffisant.

III.G L'environnement multicolonne pour simuler des tableaux

Ce n'était pas prévu dès le début du projet, mais vu sa souplesse, l'environnement multicolonne peut être utilisé pour mettre en forme des tableaux simples, mais qui peuvent être arbitrairement longs. Il aura fallu passer outre deux limitations de la première implémentation de cet environnement : gérer plus que 2 colonnes et avoir la possibilité d'ajouter un entête qui indique la nature du contenu de chaque colonne. C'est fait. De nouveaux outils ont été ajoutés pour couvrir ce besoin. En outre un système de configuration de type clé/valeur a été ajouté, le package `paracol` n'en disposant pas.

Experimental !

C'est une couche d'abstraction qui repose sur les fonctionnalités du package `paracol`. Ces outils sont encore au stade expérimental et certaines choses que peuvent faire les outils pour tableaux ne pourront jamais être imitées avec le nouvel environnement (comme la fusion de cellules par exemple). Pour plus de précaution, l'implémentation de ce nouvel environnement a été écrite à part. À terme `\bprogram` et `\bcomment` en seront de simples applications.

Entête pour le programme/commentaires

Certains outils ajoutés ont été aussi implémentés pour l'ancien environnement bicolonne comme l'insertion d'un entête ou les filets de séparations horizontaux. Mais pour le programme et ses commentaires insérer à répétition le même entête chaque fois qu'on entre en mode bicolonne n'apporterait aucune information utile au lecteur. On peut simplement expliquer au début du livret le principe des deux colonnes. Ils faudra donc réserver cette fonctionnalité aux cas où on a besoin d'un entête particulier.

III.G.1. Tutoriel

Pour une utilisation de base, deux commandes suffisent pour mettre en forme un « tableau » : `\bcolumn` et `\ecolumn`. La commande `\bcolumn` initialise l'environnement et permet ensuite de s'y déplacer horizontalement et verticalement. La commande `\ecolumn` le ferme.

La première commande `\bcolumn` qui initialise l'environnement exige un premier argument obligatoire qui indique le nombre de colonnes à créer. Les occurrences suivantes n'en ont aucun usage tant qu'on ne ferme pas l'environnement. Un exemple simpliste pourrait être de la forme

```
\bcolumn{3}      cellule 1 de la rangée 1 ...    % initialisation
\bcolumn        cellule 2 de la rangée 1 ...
\bcolumn        cellule 3 de la rangée 1 ...
\bcolumn        cellule 1 de la rangée 2 ...
\bcolumn        cellule 2 de la rangée 2 ...
\bcolumn        cellule 3 de la rangée 2 ...
\bcolumn        cellule 1 de la rangée 3 ...
\ecolumn                               % fermeture
```

code 35

Résultat du code 35 :

cellule 1 de la rangée 1 ...	cellule 2 de la rangée 1 ...	cellule 3 de la rangée 1 ...
cellule 1 de la rangée 2 ...	cellule 2 de la rangée 2 ...	cellule 3 de la rangée 2 ...
cellule 1 de la rangée 3 ...		

Sans paramètre et sans aucune indication du mouvement à suivre, `\bcolumn` se déplace de colonne en colonne sur une même rangée jusqu'à la dernière colonne et initialise ensuite d'elle même une autre rangée. Cet arrangement montre ses limites à la lecture. Difficile en effet de détecter visuellement quel contenu va dans quelle cellule au niveau des sources. Pour y remédier, dans le processus d'initialisation, d'autres commandes aux noms plus explicites sont créées. Dans notre exemple, les commandes `\cone`, `\ctwo` et `\cthree` permettent respectivement de se placer dans la première, la deuxième ou la troisième colonne. Pour créer une nouvelle rangée et se placer directement dans la colonne concernée il suffit d'ajouter un modificateur `*` à l'une de ces commandes. Par exemple `\ctwo*` crée une nouvelle rangée et se déplace vers la deuxième colonne.

L'exemple précédent pourrait ainsi être traité avec le code

```
\bcolumn{3}      cellule 1 de la rangée 1 ...
\ctwo            cellule 2 de la rangée 1 ...
\cthree          cellule 3 de la rangée 1 ...
\cone*           cellule 1 de la rangée 2 ...
\ctwo            cellule 2 de la rangée 2 ...
```

code 36

```

\cthree    cellule 3 de la rangée 2 ...
\cone*     cellule 1 de la rangée 3 ...
\ecolumn

```

code 36

Cela améliore la lecture au niveau des sources. La première commande `\bcolumn` reste obligatoire car les commandes `\cone`, `\ctwo`... restent inconnues tant que l'environnement n'a pas été initialisé. Il n'est pas nécessaire de respecter l'ordre de ces commandes sur une même rangée. Le contenu apparaîtra toujours à la bonne position. Le code suivant produit encore la même chose

```

\bcolumn{3}    cellule 1 de la rangée 1 ...
\cthree        cellule 3 de la rangée 1 ...
\ctwo          cellule 2 de la rangée 1 ...
\ctwo*         cellule 2 de la rangée 2 ...
\cthree        cellule 3 de la rangée 2 ...
\cone          cellule 1 de la rangée 2 ...
\cone*         cellule 1 de la rangée 3 ...
\ecolumn

```

code 37

Cette fois c'est la commande `\ctwo*` qui crée la deuxième rangée et on n'a pas respecté l'ordre d'entrée des cellules sur les rangées 1 et 2. Une fois une rangée créée par une commande étoilée, on peut multiplier l'appel des commandes non étoilées dans n'importe quel ordre, possiblement plusieurs fois, sans déclencher la création d'une nouvelle rangée. Si une même commande est utilisée plusieurs fois sur une rangée alors le nouveau contenu est simplement ajouté comme nouveau paragraphe (au sens \TeX) au contenu précédent sur la même colonne. Si l'une des commandes est omise sur une rangée alors le contenu de la cellule correspondante reste vide.

Pour aller plus loin dans la personnalisation des commandes de déplacement, on peut leur donner des noms plus évocateurs du contenu des colonnes. Tout doit se faire dans les options d'initialisation de l'environnement avec pour l'exemple une instruction de la forme

```

\bcolumn{3}<col names={1=prog, 2=com, 3=per}>

```

code 38

Avec celle-ci on obtient les commandes de déplacement `\bprog`, `\bcom` et `\bper` qui sont les exactes synonymes de `\cone`, `\ctwo` et `\cthree`. La lettre *b* est ajoutée au début du nom de chaque commande pour minimiser le risque de conflit avec les noms des commandes \LaTeX existantes (et pour respecter les traditions du projet). Rien n'oblige à fournir tous les noms comme dans cet exemple, les commandes de la famille `\cone` étant toujours disponibles.

Ce qu'on y gagne

Par rapport à un tableau classique on obtient visuellement une meilleure organisation logique du code source, les ruptures de pages peuvent se produire naturellement y compris au milieu d'une rangée sans perdre l'alignement vertical, les erreurs pointeront exactement là où il faut et la synchronisation

source/PDF reste opérationnelle.

Convention des options

D'autres options ont des noms qui commencent avec `col`. Elles prennent toutes une liste d'options comme valeur avec une syntaxe de la forme `{1=<opt1>,2=<opt2>,...}`. Chaque nombre représente le numéro de la colonne concernée par l'option. Il n'est pas obligatoire de fournir une option pour chaque colonne. En l'absence du numéro d'une colonne une option par défaut est adoptée.

`\bprogram` et `\bcomment`

L'implémentation des commandes originales `\bprogram` et `\bcomment` est légèrement différente. `\bprogram` crée toujours une nouvelle rangée alors que `\bcomment` ne le fait que si c'est elle qui doit intituler l'environnement bicolonne. Aucune des deux commandes n'a une version étoilée. Elles n'en ont pas besoin.

Regardons maintenant comment spécifier les largeurs des colonnes :

```
\bcolumn{4}<ratio={.25,.50}>
```

code 39

Comme pour `\bprogram` c'est l'option `ratio` qui permet de le faire mais cette fois on doit lui passer entre accolades une liste de proportions qui seront utilisées dans le même ordre pour fixer les largeurs des colonnes. Pas besoin de spécifier toutes les proportions. S'il y a plus de colonnes que d'éléments dans la liste alors ce qui reste sur la largeur de la zone de texte est divisé équitablement sur les colonnes restantes. Sur cet exemple, la première colonne sera de largeur le quart de la zone de texte, la deuxième la moitié et les deux colonnes restantes auront chacune le huitième.

Les proportions sont en fait appliquées à la largeur globale de la ligne courante diminuée de tous les espaces de séparations.

Pour l'alignement horizontal du texte dans les colonnes :

```
\bcolumn{3}<col align={1=l, 2=l, 3=c}>
```

code 40

Chaque nombre représente encore un numéro de colonne. Il reçoit en valeur un caractère unique qui indique l'alignement à appliquer : `l` (alignement à gauche), `c` (alignement au centre) ou `r` (alignement à droite). Si une colonne ne reçoit pas une directive d'alignement, elle adopte l'alignement en cours dans le texte global.

Le texte normal est justifié des deux côtés. l'option `l` donne un résultat différent : le texte est justifié seulement à gauche. Il est préférable de toujours l'activer à la place du mode normal quand la largeur de la colonne est réduite.

Maintenant le point tant attendu : un tableau ne serait pas complet s'il n'a pas d'entête avec des intitulés qui indiquent la nature de ce que contiennent les colonnes. On y arrive avec l'option « `col head` »

```
\bcolumn{3}<col head={1={l}{Programme}, 2={l}{Commentaires}, 3={c}{Période}}>
```

Pour un numéro de colonne on fournit deux arguments obligatoires entre accolades : le premier est un caractère qui indique l'alignement horizontal de l'intitulé et le deuxième est l'intitulé lui-même. Si l'argument d'alignement est vide (ie {}) alors l'alignement en cours pour toute la colonne est adopté. Les autres valeurs utilisables sont l, c et r et elles ont la même signification que pour l'alignement du texte dans les colonnes. L'entête est encadré verticalement entre deux filets d'épaisseur différentes (une convention typographique pour les tableaux) et un autre filet est inséré automatiquement à la fin du « tableau ».

Pour clore ce tutoriel, on signale qu'il est possible d'ajouter des filets de séparation horizontaux intermédiaires avec la commande `\interrule`.

Propriétés des filets

Des options sont disponibles pour régler les propriétés des filets mais pour ne pas encombrer cette partie elles seront exposées avec d'autres options de l'environnement en annexe dans la partie syntaxe complète.

III.G.2. Exemples

Maintenant des exemples suivis de leurs codes complets.

Exemple 1 : un tableau avec entête et sans filets intermédiaires. On compense avec un espace plus grand entre les rangées qu'on règle avec l'option `interskip factor`.

Programme	Commentaires	Période
Contenu du programme dans la première cellule.	Son commentaire n'as pas grand chose à dire.	Période.
Encore du texte mais sans commentaires svp. Ce sont les vacances quand même.		Vacances
	Pas besoin de texte pour faire un commentaire.	Inconnue

```
\bcolumn{3}<
  ratio={.4,.4},
  col names={1=prog, 2=com, 3=per},
  col align={3=c},
  col head={1={{Programme}}, 2={{Commentaires}}, 3={{Période}}},
```

code 42

```

interskip factor=2,
>
    Contenu du programme dans la première cellule
\bcom  Son commentaire n'as pas grand chose à dire
\bper  Période
\bprog* Encore du texte mais sans commentaires svp. Ce sont les vacances quand m
ême.
\bper  Vacances
\bcom*  Pas besoin de texte pour faire un commentaire
\bper  Inconnue
\ecolumn

```

Exemple 2 : le même avec filets de séparation horizontaux et verticaux

Programme	Commentaires	Période
Contenu du programme dans la première cellule	Son commentaire n'as pas grand chose à dire à part remplir le vide.	Période
Encore du texte mais sans commentaires svp. Ce sont les vacances quand même.		Vacances
	Pas besoin de texte pour faire un commentaire	Inconnue

code 43

```

\bcolum{3}<
    ratio={.4,.4},
    col names={1=program, 2=comment, 3=periode},
    col align={3=c},
    col head={1={{Programme}}, 2={{Commentaires}}, 3={{Période}}},
    seprule,
>
    Contenu du programme dans la première cellule
\bcom  Son commentaire n'as pas grand chose à dire
\bper  Période
\interrule
\bprog* Encore du texte mais sans commentaires svp. Ce sont les vacances quand m
ême.
\bper  Vacances

```

code 43

```
\interrule
\bcom*   Pas besoin de texte pour faire un commentaire
\bper    Inconnue
\ecolumn
```

Exemple 3 : et un dernier montrant comment créer un alignement vertical en utilisant la commande `\vfill`

Compression

Un paragraphe assez long pour le faire étaler sur plusieurs lignes dans une colonne très étroite.

Dilatation

Texte court centré verticalement

code 44

```
\bcolumn{2}<
  ratio={.3},
  seprule,
  col align={1=l,2=c},
  col head={1={{Compression}}, 2={{Dilatation}}}
>
  Un paragraphe assez long pour le faire étaler sur plusieurs lignes dans une
  colonne très étroite.
\bcolumn  \vfill Texte court centré verticalement \vfill
\ecolumn
```

Flottement vertical

Sur une même rangée, le texte de chaque colonne commence verticalement au même niveau. Quoiqu'il soit possible d'imposer un placement vertical du texte dans une « cellule » aux trois positions possibles, en haut, au centre ou en bas, il n'a pas été jugé utile de l'ajouter comme fonctionnalité native de l'environnement. Un tel alignement est facile à simuler en plaçant convenablement la commande primitive `\vfill` avant et/ou après le contenu de la cellule.

III.G.3. Syntaxe Complète

```
\bcolumn{<int><keyval>[<int>][<code>]
\bcolumn[<int>]*[<code>]
  \col[<int>]*[<code>]
\ecolumn
```

Dans la première syntaxe, la première instance de `\bcolumn`, initialise l'environnement multicolonne avec `<int>` colonnes et se redéfinit elle-même pour qu'elle devienne une commande de déplacement dans l'environnement. Une fois l'environnement fermé par `\ecolumn`, elle reprend son rôle d'ini-

tialisatrice. Si on utilise le paramètre {<int>} alors que l'environnement est déjà ouvert alors <int>, tout comme les « options » suivantes, sont simplement intégrés au flux de texte normal.

Une fois l'environnement initialisé avec \bcolumn la commande \bcolumn devient synonyme de la commande au nom plus court \col. Celle-ci reste inconnue tant que l'environnement n'a pas été ouvert. Une autre famille de commande est aussi créée dynamiquement à l'initialisation : \cone, \ctwo, \cthree... qui sont respectivement synonymes de \col[1], \col[2], \col[3]...

- {<int>} <int> comme on l'a vu, il indique le nombre de colonnes ;
- <keyval> pour les options de l'environnement. Si ce paramètre est utilisé alors que l'environnement est déjà ouvert, il est considéré comme du texte normal et ajouté au texte qui suit la commande ;
- [<int>] <int> indique la colonne dans laquelle le texte suivant va être placé. Si ce paramètre est absent alors la colonne suivante est sélectionnée et dans le cas où on se trouve déjà dans la dernière colonne d'une rangée, une nouvelle rangée est créée et la première colonne est sélectionnée ;
- * avec ce paramètre optionnel on impose la création d'une nouvelle rangée. Si [<int>] a été utilisé avant alors on se déplace directement dans la <int>^{ème} colonne. S'il est absent on se place dans la première colonne ;
- [<code>] comme pour \bprogram, le code <code> est placé en pleine largeur tout en se plaçant dans la colonne <int> pour le texte suivant ;

Une fois l'environnement initialisé, la commande \bcolumn (ou \col) permet d'y faire tout genre de déplacement. Elle a un comportement plus primitif et pas facile à cerner :

- comme on l'a vu, sans paramètre, elle se déplace de colonne en colonne et crée une nouvelle rangée une fois la rangée en cours épuisée. Sans le paramètre [<int>] elle ne provoque jamais d'erreur ;
- avec l'instruction \bcolumn[<int>] elle se déplace directement dans la <i>^{ème} colonne. Une erreur est déclenchée si <i> est en dehors des limites du tableau ;
- avec l'instruction \bcolumn[+<i>] elle se décale de <i> colonne(s) vers la droite à partir de la position courante. Elle provoque une erreur si on déborde des limites du tableau.
- avec l'une des instructions \bcolumn[<i>]* ou \bcolumn[+<i>]* elle crée d'abord une nouvelle rangée et exécute le mouvement demandé.

En fait, les commandes de la famille \cone utilisent en interne \col. La commande \cfour est par exemple exactement la même chose que \col[4].

Il peut être toutefois préférable, pour plus de clarté du code source, d'utiliser les commandes de la famille \cone, ou bien des noms personnalisés (voir le tutoriel précédent).

La liste, non exhaustive ⁴, des options utilisables dans <keyval> :

4. Les noms et les effets des options ne sont pas encore définitivement arrêtés

ratio={ $\langle list \rangle$ }

$\langle list \rangle$ est une liste de $\langle num \rangle$ qui indiquent dans l'ordre les proportions de largeur qu'auront successivement les colonnes. Le ratio est appliqué à la largeur globale de la zone de texte moins la somme des espaces de séparations. S'il y a moins de nombres dans la liste que de colonnes alors les colonnes restantes se partageront l'espace qui reste sur la ligne.

col widths={ $\langle list \rangle$ }

Une version bien plus polyvalente que l'option **ratio**. $\langle list \rangle$ est une liste d'options de la forme :

$$\langle int \rangle = \{ \langle dim1 \rangle \} \{ \langle dim2 \rangle \}$$

Les dimensions $\langle dim1 \rangle, \langle dim2 \rangle$ désignent la largeur de la colonne $\langle i \rangle$ et celle de l'espace de séparation qui la suit. Comme pour toutes les options «col ...», il n'est pas nécessaire de fournir les spécifications de toutes les colonnes. Dans ce cas les colonnes absentes se partagent l'espace restant sur la ligne. Si l'un des deux arguments $\{ \langle dim1 \rangle \}$ ou $\{ \langle dim2 \rangle \}$ est vide alors une valeur par défaut est adoptée : la colonne $\langle i \rangle$ recevra sa part de ce qui reste sur la colonne pour le premier et sera suivi de l'espace de séparation commun pour le deuxième. Il est possible aussi de spécifier les deux dimensions en terme de proportions. Dans ce cas elles doivent être sous la forme $* \langle num1 \rangle$ et/ou $* \langle num2 \rangle$ (sans unité). Pour $\langle num1 \rangle$ la colonne sera de largeur égale à la largeur commune des colonnes sans spécification multipliée par le facteur $\langle num1 \rangle$. Pour $\langle num2 \rangle$, l'espace de séparation sera celui normal multiplié par le facteur $\langle num2 \rangle$.

Exemple : sur un tableau à 4 colonnes, avec l'option

$$\text{col widths}=\{1=\{*2\}\}, 3=\{1\text{cm}\}\{*.5\}\}$$

on obtient : la troisième colonne avec une largeur de 1cm et l'espace de séparation qui la suit égale la moitié de celui normal ; les colonnes 2 et 4 avec la même largeur et la colonne 1 deux fois cette dernière ; les autres espaces de séparations sont égaux à l'espace normal.

Voilà ! Toutes les spécifications possibles, et même plus, avec un tableau \LaTeX créé avec le package **tabularx** sont là. Merci **paracol**.

col names={ $\langle list \rangle$ }

$\langle list \rangle$ est une liste d'options de la forme $\langle int \rangle = \langle csname \rangle$. Le nombre $\langle int \rangle$ représente le numéro d'une colonne et $\langle csname \rangle$ est utilisé pour créer la commande $\text{\backslash b}\langle csname \rangle$ dont la signification est équivalente à $\text{\backslash col}[\langle int \rangle]$. Une erreur est déclenchée si $\langle int \rangle$ est au delà des limites du tableau.

col styles={ $\langle list \rangle$ }

$\langle list \rangle$ est ici une liste d'options de la forme $\langle int \rangle = \langle code \rangle$, le code \LaTeX $\langle code \rangle$ est alors exécuté au début de chaque cellule sur la $\langle i \rangle^{\text{ème}}$ colonne. Il permettra par exemple d'appliquer un style de police différent pour la colonne.

col align={ $\langle list \rangle$ }

cette fois $\langle list \rangle$ est une liste d'options de la forme $\langle int \rangle = \langle char \rangle$ avec $\langle char \rangle$ qui indique l'alignement horizontal du texte dans la colonne. Les valeurs possibles sont **r**, **c** ou **l**.

col head={ $\langle list \rangle$ }

$\langle list \rangle$ est une liste d'options de la forme $\langle int \rangle = \{ \langle char \rangle \} \{ \langle text \rangle \}$ où $\langle text \rangle$ sert comme intitulé de la colonne dans l'entête du tableau et $\langle char \rangle$ indique l'alignement de celui-ci. Si $\{ \langle char \rangle \}$ est vide alors l'alignement en cours pour la colonne est utilisé. La présence de l'option `col head` active automatiquement l'insertion de l'entête au début et un filet de terminaison à la fin du tableau.

seprule = $\{ \langle dim \rangle \} \{ \langle color \rangle \}$ déf : $\{ 0.4pt \} \{ strokecol \}$

active les filets de séparations verticaux entre toutes les colonnes et réglent leurs propriétés. $\langle dim \rangle$ est l'épaisseur du filet, sa valeur par défaut est selon le standard L^AT_EX 0.4pt. $\langle color \rangle$ est le nom d'une couleur qui sera utilisée pour le filet. Sa valeur par défaut est `strokecol` qui est le nom interne d'une couleur qui est utilisée pour tout type de filet. Si l'option `seprule` est utilisée sans valeur alors les filets sont activés et ils utilisent les valeurs par défaut.

III.G.4. Outils annexes

\interskip $\{ \langle frac \rangle \}$

Permet, alors qu'on est en mode multicolonne, d'insérer correctement un saut vertical élastique entre deux rangées. Ce saut est de valeur égale à `\parskip` multiplié par le fraction $\langle frac \rangle$ fournie en argument.

\interrule $\langle keyval \rangle$

Permet de tracer un filet horizontal sur toute la largeur de la zone de texte alors qu'on se trouve dans un contexte multicolonne et sans quitter celui-ci. On peut changer les propriétés du filet en utilisant le paramètre $\langle \rangle$ avec les options suivantes :

h = $\langle dim \rangle$ déf : 0.4pt

fixe l'épaisseur du filet. Sa valeur par défaut est de 0.4pt (h pour height);

c = $\langle color \rangle$ déf : 

le nom d'une couleur qui sera utilisée pour tracer le filet;

b = $\langle dim \rangle$ déf : 3pt

fixe l'espace vertical non résiliable avant le filet (b pour before).

a = $\langle dim \rangle$ déf : 3pt

la même chose mais pour l'espace vertical après le filet (a pour after).

Sans l'argument optionnel la commande `\interrule` revient ainsi à l'instruction :

`\interrule<h=0.4pt, a=3pt, b=3pt>`

Les commandes `\interskip` et `\interrule` déclenchent des erreurs si elles sont utilisées en dehors d'un contexte multicolonne.

III.H Notes de bas de page et éléments flottants

Dans cette version les notes de bas de page (`\footnote`) sont insérées de manière normale, en bas de la page en cours

Dans les versions précédentes elles étaient insérées en bas de chaque colonne, mais seulement lorsque l'environnement multicolonne est fermé. Éventuellement sur une autre page.

III.I Recommandations pour la définition de nouvelles commandes

Dans le cas où un fichier de base utilise des commandes ou des extensions non prévues par les fichiers du projet, il suffit de créer un fichier qui porte le même nom que le fichier de base avec l'extension `.def` à côté du fichier maître générique et qui contient le nécessaire. Insérer ensuite (exceptionnellement) une ligne pour son inclusion dans le *préambule* du fichier maître (avant `\begin{document}`).

Par exemple : `\input chi-1e-mp---.def`. Sachant que `chi-1e-mp---.def` contiendrait quelque chose comme

```
\usepackage{chemarrow}
\usepackage{chemexec}
\newcommand{\dd}[2]{\frac{\partial #1}{\partial #2}}
. . .
```

code 45

Le cas échéant une compilation de ces fichiers sera préparée pour la production des livrets.

IV Construction d'un fichier maître

IV.A Options de `cpgelvrt`

l'utilisation de `cpgelvrt.cls`, une fois qu'il est mis à portée du compilateur⁵ s'utilise en insérant en début du fichier maître une ligne de la forme

```
\documentclass[options]{cpgelvrt}
```

Les options reconnues se divisent en deux catégories. Celles en relation avec la présentation et celle qui indiquent le type de livret à produire.

Dans la deuxième catégorie, on dispose des options :

- generic** pour un fichier maître générique (type 00). Ce type de fichier ne sert que dans la phase de rédaction ou de traitement d'un fichier de base ;
- eleve** pour la production d'un livret à destination des élèves. Il contient les programmes de toutes les matières pour une filière et un niveau donnés. Ne contient pas de parties (`\insertpart`) ;
- prof** pour la production d'un livret à destination des professeurs (type 11). Il contient tous les programmes pour une matière et une filière donnés. Ne contient pas de parties (`\insertpart`) ;

5. à côté du fichier à compiler par exemple

- inspec** pour la production d'un livret à destination des inspecteurs (type 12). Il contient tous les programmes d'une matière, pour toutes les filières et tous les niveaux. Les parties sont constituées des programmes pour une filière ;
- admin** pour la production d'un livret à destination des administrations (type 21). Il contient tous les programmes pour une filière donnée, toutes matières et tous niveaux confondus. Les parties sont constitués des programmes pour un même niveau ;
- autre** pour les livrets destinés aux autres partis concernés par les nouveaux programmes des CPGE (type 22).
- other** option qui permet d'utiliser `cpge1vrt.cls` pour produire autre chose que les livrets des programme (ce manuel par exemple) tout en appliquant le style du projet. Elle désactive les routines de traitement des informations relatives aux fichiers des programmes.

Chaque option applique les directives générales propres au type de fichier maître pour la gestion des parties, la construction des titres et éventuellement la production des couvertures. (voir aussi la commande `\setinfo` plus loin). L'option par défaut est `generic`.

Les autres options sont

- NB** pour produire un document en niveau de gris. Les livrets sont par défaut produit en couleurs conformément à la charte graphiques du ministère ;
- height**= $\langle dim \rangle$ déf : 24cm
option facultative pour indiquer la hauteur du format à retenir pour le livret. La valeur par défaut, en cas de non utilisation de l'option est 24cm ;
- width**= $\langle dim \rangle$ déf : 19cm
la même chose mais pour la largeur du livret ;
- cover** option pour la production de la couverture seulement ;
- coversep**= $\langle dim \rangle$ déf : 0mm
mesure de réserve qui s'ajoute au double de la largeur du livret pour constituer la largeur de la couverture.

IV.B Commandes principales

Trois commandes assurent par un système clé/valeur la passation des informations concernant le fichier en cour de traitement. Il s'agit des commandes `\setinfo`, `\insertpart` et `\includebase`. Chacune remplit un rôle particulier qui est décrit dans la suite.

Les clés sont communes aux trois commandes : `filieres`, `niveau` et `matiere`. Leurs noms sont évocateurs de leurs significations. Le tableau suivant indique les valeurs qui peuvent leurs être affectées.

Clé	Valeurs possibles
filiere	mp, psi, pc, tsi, ecs, ect
niveau	1 ou 2
matiere	mat, phy, chi, sci, gee, gem, egm, eed, ehg, fra, ara, ang

• sci = sciences de l'ingénieur; • gee = génie électrique; • gem = génie mécanique; • egm = sciences de gestion et management; • eed = économie-droit et • ehg = histoire, géographie et géopolitique.

Il est indispensable de respecter la syntaxe des clés et de leurs valeurs sous peine d'erreur de compilation.

La syntaxe de ces commandes, dont le résultat dépend dynamiquement de l'option de la classe `cpgelvrt` précisant le type de livret à produire (`generic`, `eleve`, `prof` ...), est de la forme

`\setinfo{<options>}`

`\insertpart[<options>]`

`\includebase[<options>]{<fichier>}`

`<options>` est ici une ou plusieurs entrées de la forme `clé=valeur` utilisant les couples du tableau précédent et séparées par des virgules.

`\setinfo`

n'a d'autre fonction que d'affecter des valeurs à l'une ou à plusieurs des variables `filiere`, `niveau` et `matiere`.

Ce qui permet par exemple, juste après le `\begin{document}`, d'entrer les informations nécessaires à la génération du titre du livret avec `\maketitle`. Elle est utilisable partout dans le document pour éventuellement compléter une information manquante. Les valeurs affectées aux variables le sont de façon permanente et ne peuvent être résiliées que par une autre commande `\setinfo`. Comme les commandes `\insertpart` et `\includebase` utilisent elles même `\setinfo`, leur utilisation avec des paramètres optionnels revient à changer de façon permanente les valeurs des clés qu'elle modifie.

Par exemple, lorsque on traite le fichier de type 12 pour les mathématiques

```
\setinfo{matiere=mat}
\maketitle
```

code 46

La commande `\setinfo` renseigne la clé `matiere`. Ce qui est suffisant pour générer la page de garde avec `\maketitle` pour ce livret.

`\insertpart`

insère le titre d'une partie en utilisant éventuellement les informations fournies ou déjà disponibles.

Ex. : pour le même fichier de type 12, la première partie est introduite par

`\insertpart[filiere=mp]`, la deuxième par `\insertpart[filiere=psi]` ...

\includebase

est responsable de l'inclusion d'un fichier de base tout en fournissant les informations nécessaires (manquantes). Le paramètre *<fichier>* est le nom du fichier de base à inclure (sans l'extension .tex et sans indication du chemin, juste le nom du fichier lui même).

Ex. : toujours pour le même fichier de type 12, et dans la première partie

```
\includebase[niveau=1]{mat-1e-mpsi-}
```

```
\includebase[niveau=2]{mat-2e-mp---}
```

IV.B.1. Exemple :

Contenu du fichier 11-prg-chi-mp---.tex

```
\documentclass[prof]{cpgelvrt}
\begin{document}
\setinfo{filier=mp,matiere=chi}
\maketitle
\includebase[niveau=1]{chi-1e-mpsi-}
\includebase[niveau=2]{chi-2e-mp---}
\tableofcontents
\end{document}
```

code 47

IV.C Cr ation d'une couverture

Il n'y a pour cela, pas grand chose   faire. Placer l'option cover et celle du type de fichier, utiliser `\setinfo` pour renseigner les diff rentes informations exigibles et placer ensuite une commande `\maketitle`.

Les fichiers des couvertures devraient  tre cr  s s par ment des fichiers ma tre. Un fichier de couverture pour chaque fichier ma tre. Son nom est d duit du fichier maitre en rempla ant prg par cov.

IV.C.1. Exemple

Contenu du fichier 11-cov-chi-mp---.tex

```
\documentclass[prof,cover,coversep=6mm,decolength=58,fonts]{cpgelvrt}
\begin{document}
\setinfo{matiere=bio,filier=bcpst}
\maketitle
\end{document}
```

code 48

V Informations supplémentaires

V.A Mise en page

`cpge\lvt` invoque l'extension `geometry` pour régler les détails de la mise en page. Les différentes mesures de la mise en page sont calculées automatiquement à partir des seules valeurs affectées aux clés `width` et `height` lors du chargement de la classe. Valeurs qui désignent respectivement la largeur et la hauteur du papier à utiliser.

Par exemple

```
\documentclass[width=19cm,height=24cm, ...]{cpge\lvt}
```

code 49

En cas de non utilisation des clés `width` et `height` les valeurs par défauts 19cm et 24cm leurs sont affectées.

V.B Polices de caractères

Pour les polices de caractères, les commandes suivantes sont prévues : `\CommentFont`, `\TitlingFont`, `\HeadFont`, `\BlackFont`, `\ObjFont`, `\ContentsFont`. Ce sont des macros commandes \LaTeX et on peut les redéfinir en utilisant `\renewcommand`. Normalement elles sont définies dans les fichiers `gtdfont.sty` et `gtdxfont.sty`, mais pour une utilisation générique elles sont définies comme suit :

```
\newcommand\ObjFont{\sffamily\itshape}
  police utilisée dans les environnements objectif et objectif*;
```

```
\newcommand\TitlingFont{\sffamily}
\newcommand\BlackFont{\sffamily\bfseries}
  polices utilisées dans les titres;
```

```
\newcommand\HeadFont{\sffamily\scshape}
  police utilisée dans les entêtes et pieds de pages.
```

V.C Couleur

Les documents produits, le sont par défaut en couleur. La mise en forme et la palette des couleurs est conforme à la charte graphique du Ministère (Voir tableau TAB. V.2 page 55). Les couleurs sont encodées en CMJN.

Néanmoins on peut facilement convertir l'espace des couleurs en niveau de gris avec la commande standard (de l'extension `xcolor`) : `\selectcolormodel{gray}`

Ce qui aura l'avantage de conserver les tonalités des couleurs de la charte. L'option `NB` de la classe de document `cpge\lvt` permet de produire le document en niveau de gris en appliquant en interne la commande précédente .










Nom	Couleur	définition
cgbluetext		<code>\definecolor{cgbluetext}{cmyk}{1,.9,.1,0}</code>
cgbluefill		<code>\definecolor{cgbluefill}{cmyk}{1,.64,0,.6}</code>
cgorangetext		<code>\definecolor{cgorangetext}{cmyk}{0,.7,1,0}</code>
cgorangestroke		<code>\definecolor{cgorangestroke}{cmyk}{0,.4,.9,0}</code>
cgorangefill		<code>\definecolor{cgorangefill}{cmyk}{0,.28,.76,0}</code>
cggold1		<code>\definecolor{cggold1}{cmyk}{.07,.27,.94,.16}</code>
cggold2		<code>\definecolor{cggold2}{cmyk}{.29,.3,.62,.04}</code>
cggraytext		<code>\colorlet{cggrayfill}{black !20}</code>
cggrayfill		<code>\colorlet{cggraytext}{black !70}</code>

TABLE V.2 – Les couleurs de la charte avec leurs définitions

V.D Extensions utilisées par les fichiers styles

Extension	utilité
<code>xkeyval</code> ⁶ , <code>pgfkeys</code> ⁷ , <code>xstring</code> ⁸ , <code>etoolbox</code>	pour l'aspect programmation.
<code>xcolor</code>	gestion avancée de la couleur.
<code>hyperref</code>	pour tout ce qui touche aux liens hypertexte et aux bookmark PDF
<code>tikz</code>	pour l'ajout d'éléments graphiques à la présentation.
<code>paracol</code>	Peut être l'extension la plus importante utilisée par le projet. C'est elle qui rend possible la présentation sous formes de colonnes le texte du programme et les commentaires associés.
<code>nccparskip</code> et <code>setspace</code>	petites extensions pour modifier correctement le sauts de paragraphe et interligne.
<code>titlesec</code> et <code>titletoc</code>	pour la personnalisation des titres, des entêtes/pieds de page et de la table des matières.

<code>enumitem</code>	pour la personnalisation des listes
<code>eso-pic</code>	pour l'insertion de la bande grise sur les côtés extérieurs de toutes les pages du document.
<code>flowframe</code>	extension qui permet de subdiviser la page en plusieurs frames. Utilisée dans la mise en page des couvertures.

TABLE V.3 – Les extensions utilisées par les fichiers styles du projet

-
- 6. `xkeyval` est responsable de la gestion des options de la classe. Une migration vers `pgfkeys` est prévue
 - 7. `pgfkeys` est responsable du système clé/valeur pour les fichiers de base.
 - 8. `xstring` est à la base des mots clés pour le style des compteurs

Table des matières

Manuel de l'utilisateur	Page	2
I Historique des versions		3
II Installation et normalisation		5
A Installation des fichiers du projet		5
B Fichiers style du projet		7
C Conventions de nomenclature des fichiers		9
Exemples		10
D Contenu des fichiers		10
E Normalisation des fichiers de bases		10
F À propos de typographie		11
G Rigide ou élastique		13
H Plaidoyer pour les solutions de multicolonnage		15
III Traitement des fichiers de base		16
A Conventions de notation		16
B Titres de sections et table des matières		18
Sections et table des matières générales		19
La commande \subchapter		20
Titres des sessions de TP		20
Styles des titres		21
Tables des matières partielles		24
C Objectifs et ordre		24
D Programme et commentaires		25
Les commandes principales		25
Outils annexes		29
Marquage des paragraphes du programme		29
E Des titres moins contraignants		30
F Des listes plus expressives		34
Tutoriel		34
La syntaxe		38

G	L'environnement multicolonne pour simuler des tableaux	40
	<i>Tutoriel</i>	41
	<i>Exemples</i>	44
	<i>Syntaxe Complète</i>	46
	<i>Outils annexes</i>	49
H	Notes de bas de page et éléments flottants	49
I	Recommandations pour la définition de nouvelles commandes	50
IV	Construction d'un fichier maître	50
A	Options de cpgelvrt	50
B	Commandes principales	51
	<i>Exemple :</i>	53
C	Création d'une couverture	53
	<i>Exemple</i>	53
V	Informations supplémentaires	54
A	Mise en page	54
B	Polices de caractères	54
C	Couleur	54
D	Extensions utilisées par les fichiers styles	55