

Une courte (?) introduction à ConTEXt Mark IV

Une courte (?) introduction à ConTeXt Mark IV

Version 1.6 [14 juin 2021]

© 2020-2021, Joaquín Ataz-López

Titre original : Una introducción (no demasiado breve) a ConTeXt Mark IV

Traduction française : A bon ami qui souhaite rester anonyme.

L'auteur du présent texte, ainsi que ses traducteurs anglais et français, autorisent sa libre distribution et utilisation, ce qui inclue le droit de copier et de redistribuer ce document sur support numérique à condition que l'auteur soit cité, et que le document ne soit inclus ni dans un paquet, ni dans une suite logicielle, ni dans une documentation dont les conditions d'utilisation ou de distribution ne prévoient pas la le droit de copie et de redistribution libre par ses destinataires. La traduction du document est également autorisée, à condition que la paternité du texte original soit indiquée, que son statut de traduction soit indiqué, et que le texte traduit soit distribué sous la licence FDL de la Fondation pour le Logiciel Libre (Free Software Foundation), une licence Creative Commons qui autorise la copie et la redistribution, ou autre licence similaire.

Nonobstant ce qui précède, la publication et la commercialisation de ce document, ou sa traduction, nécessitera l'autorisation écrite expresse de l'auteur.

Historique des versions :

- 18 août 2020 : Version 1.0 (Uniquement en espagnol) : Document original.
- 23 août 2020 : Version 1.1 (Uniquement en espagnol) : Correction de petites erreurs de frappe et de malentendus de l'auteur.
- 3 septembre 2020 : Version 1.15 (Uniquement en espagnol) : Autres corrections de petites erreurs de frappe et de malentendus.
- 5 septembre 2020 : Version 1.16 (Uniquement en espagnol) : Autres corrections de petites erreurs de frappe et de malentendus ainsi que des petites modifications qui améliorent la clarté du texte (je crois).
- 6 septembre 2020 : Version 1.17 (Uniquement en espagnol) : C'est incroyable le nombre de petites erreurs que je trouve. Si je veux m'arrêter, je dois arrêter de relire le document.
- 21 octobre 2020 : Version 1.5 (Uniquement en espagnol) : Introduction de suggestions et correction des erreurs signalées par les utilisateurs de la liste de diffusion [ntg-context](#).
- 14 juin 2021 : Version 1.6 : Corrections suggérées après une nouvelle lecture du document, à l'occasion de sa traduction en anglais

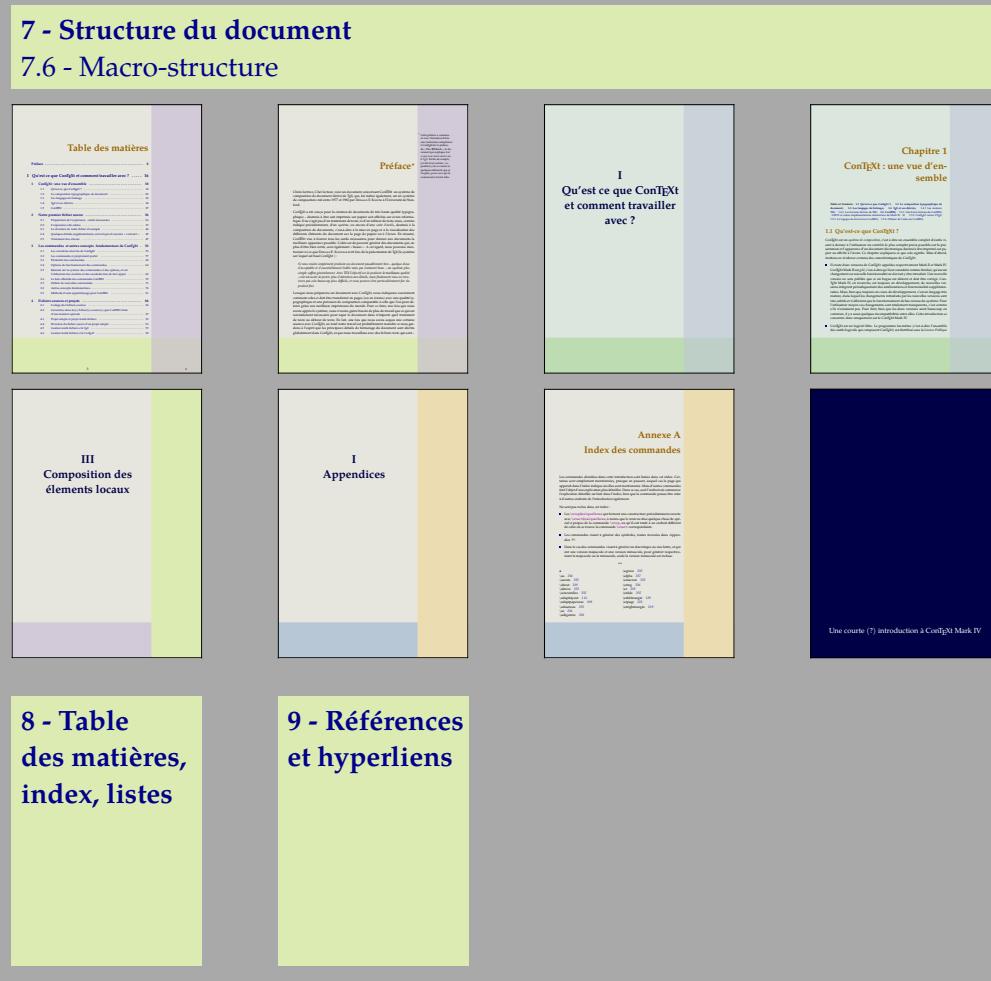
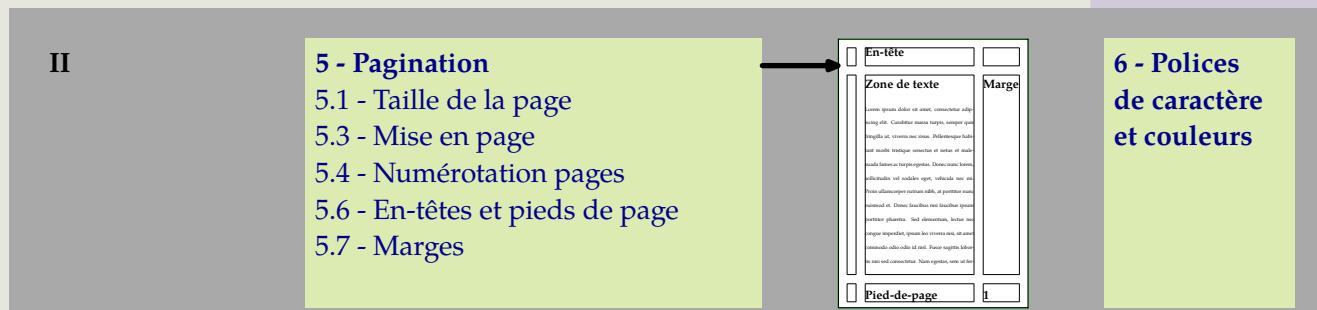
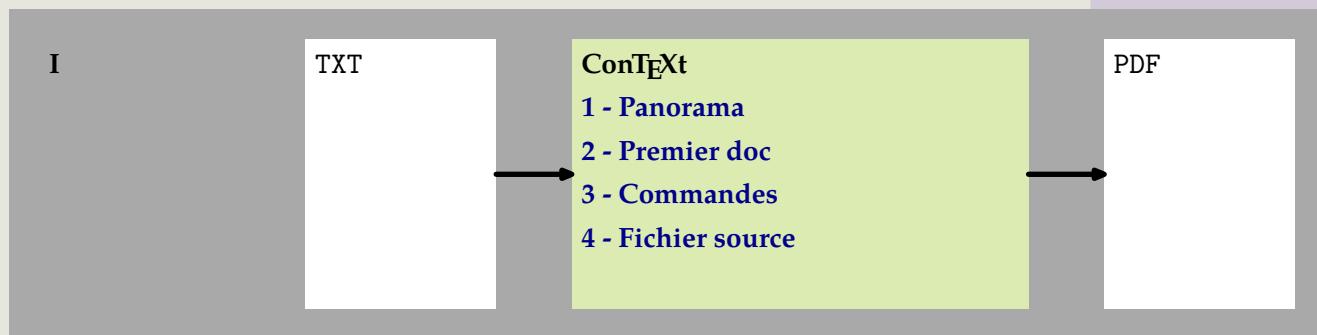
Table des matières

Table des matières	3
Préface	8
I Qu'est ce que ConTeXt et comment travailler avec ?	16
1 ConTeXt : une vue d'ensemble	18
1.1 Qu'est-ce que ConTeXt ?	18
1.2 La composition typographique de document	20
1.3 Les langages de balisage	22
1.4 TeX et ses dérivés	24
1.5 ConTeXt	27
2 Notre premier fichier source	36
2.1 Préparation de l'expérience outils nécessaires	36
2.2 L'expérience elle-même	39
2.3 La structure de notre fichier d'exemple	44
2.4 Quelques détails supplémentaires sur la façon d'exécuter « context » ..	45
2.5 Traitement des erreurs	47
3 Les commandes et autres concepts fondamentaux de ConTeXt .	52
3.1 Les caractères réservés de ConTeXt	53
3.2 Les commandes à proprement parler	57
3.3 Périmètre des commandes	60
3.4 Options de fonctionnement des commandes	64
3.5 Résumé sur la syntaxe des commandes et des options, et sur l'utilisation des crochets et des accolades lors de leur appel.	68
3.6 La liste officielle des commandes ConTeXt	70
3.7 Définir de nouvelles commandes	71
3.8 Autres concepts fondamentaux	76
3.9 Méthode d'auto apprentissage pour ConTeXt	80
4 Fichiers sources et projets	82

4.1	Codage des fichiers sources	82
4.2	Caractères dans le(s) fichier(s) source(s) que ConTeXt traite d'une manière spéciale	85
4.3	Projet simple et projet multi-fichiers	90
4.4	Structure du fichier source d'un projet simple	91
4.5	Gestion multi-fichiers <i>à la T_EX</i>	93
4.6	Gestion multi-fichiers <i>à la ConTeXt</i>	96
II	Composition des éléments généraux au document ...	102
5	Pages et pagination	104
5.1	Taille de la page	104
5.2	Éléments sur la page	109
5.3	Mise en page (<code>\setuplayout</code>)	112
5.4	Numérotation des pages	117
5.5	Sauts de page forcés ou suggérés	120
5.6	En-têtes et pieds de page	122
5.7	Insertion d'éléments de texte dans les bords de page et les marges ...	127
6	Polices d'écriture et couleurs dans ConTeXt	130
6.1	Polices de caractères incluses dans « ConTeXt Standalone »	130
6.2	Caractéristiques d'une police	132
6.3	Définition de la police principale du document	135
6.4	Modification de la police ou de certaines de ses caractéristiques	139
6.5	Autres questions relatives à l'utilisation de styles alternatifs	146
6.6	Utilisation et configuration des couleurs	148
6.7	Bonus 1 - Utilisation des polices du système d'exploitation	154
7	Structure du document	160
7.1	Les divisions structurelles d'un document	160
7.2	Types et hiérarchie des sections	162
7.3	Syntaxe commune des commandes liées aux sections	164
7.4	Format et configuration des sections et de leurs titres	166
7.5	Définir de nouvelles commandes de section	182
7.6	La macrostructure du document	183
8	Table des matières, index, listes	186
8.1	Table des matières	186
8.2	Listes, listes combinées et tables des matières basées sur une liste ...	201
8.3	Index	206
9	Références et hyperliens	212
9.1	Types de référence	212
9.2	Références internes	214
9.3	Documents électroniques interactifs	222
9.4	Hyperliens vers des documents externes	224

9.5	Création de signets dans le PDF final	228
9.6	Pièces jointes	229
9.7	Références bibliographiques	230
III	Composition des éléments locaux	232
10	Caractères, mots, texte et espace horizontal	234
10.1	Obtenir des caractères qui ne sont pas normalement accessibles à partir du clavier	234
10.2	Formatages spéciaux des caractères	243
10.3	Espacement des caractères et des mots	248
10.4	Mots composés	253
10.5	La langue du texte	255
11	Paragraphes, lignes et espace vertical	266
11.1	Les paragraphes et leurs caractéristiques	266
11.2	Espace vertical entre les paragraphes	271
11.3	Comment ConTeXt construit les lignes qui forment les paragraphes ...	277
11.4	Espace interligne	283
11.5	Autres questions relatives aux lignes	286
11.6	Alignement horizontal et vertical	289
12	Constructions et paragraphes spéciaux	294
12.1	Notes de bas de page et de fin de document	294
12.2	Paragraphes avec plusieurs colonnes	306
12.3	Listes structurées	311
12.4	Descriptions et énumérations	320
12.5	Lignes et cadres	326
12.6	Autres environnements et constructions d'intérêt	338
13	Images, tableaux et autres objets flottants	342
13.1	Que sont les objets flottants et que font-ils ?	343
13.2	Images externes	345
13.3	Tableaux	357
13.4	Aspects communs aux images, tableaux et autres objets flottants	375
13.5	Définition d'objets flottants supplémentaires	381
I	Appendices	384
A	Installing, configuring and updating ConTeXt	386
1	Installing and configuring « ConTeXt Standalone »	387
2	Installing LMTX	392
3	Using several versions of ConTeXt on the same system (only for Unix-type systems)	396

B	Commandes pour générer des symboles mathématiques et non mathématiques	398
C	Index des commandes	402
D	Index	410



Préface*

* Cette préface a commencé avec l'intention d'être une traduction/adaptation à ConTEXt de la préface de « The T_EXBook », le document qui explique *tout ce que vous devez savoir sur le T_EX*. En fin de compte, j'ai dû m'en écarter ; cependant, j'en ai conservé quelques éléments qui, je l'espère, pour ceux qui le connaissent, feront écho.

Chère lectrice, Cher lecteur, voici un document concernant ConTEXt un système de composition de document dérivé de T_EX, qui, lui même également, est un système de composition créé entre 1977 et 1982 par DONALD E. KNUTH à l'Université de Stanford.

ConTEXt a été conçu pour la création de documents de très haute qualité typographique – destinés à être soit imprimés sur papier soit affichés sur écran informatique. Il ne s'agit pas d'un traitement de texte, ni d'un éditeur de texte, mais, comme indiqué précédemment, d'un *système*, ou encore d'une *suite d'outils*, destinés à la composition de documents, c'est-à-dire à la mise en page et à la visualisation des différents éléments du document sur la page de papier ou à l'écran. En résumé, ConTEXt vise à fournir tous les outils nécessaires pour donner aux documents la meilleure apparence possible. L'idée est de pouvoir générer des documents qui, en plus d'être bien écrits, sont également « beaux ». A cet égard, nous pouvons mentionner ici ce que DONALD E. KNUTH a écrit lors de la présentation de T_EX (le système sur lequel est basé ConTEXt) :

Si vous voulez simplement produire un document passablement bon – quelque chose d'acceptable et d'essentiellement lisible mais pas vraiment beau – un système plus simple suffira généralement. Avec T_EX l'objectif est de produire la meilleure qualité ; cela nécessite de porter plus d'attention aux détails, mais finalement vous ne trouverez pas cela beaucoup plus difficile, et vous pourrez être particulièrement fier du produit fini.

Lorsque nous préparons un document avec ConTEXt, nous indiquons exactement comment celui-ci doit être transformé en pages (ou en écrans) avec une qualité typographique et une précision de composition comparable à celle que l'on peut obtenir grâce aux meilleurs imprimeurs du monde. Pour ce faire, une fois que nous avons appris le système, nous n'avons guère besoin de plus de travail que ce qui est normalement nécessaire pour taper le document dans n'importe quel traitement de texte ou éditeur de texte. En fait, une fois que nous avons acquis une certaine aisance avec ConTEXt, au total notre travail est probablement moindre si nous gardons à l'esprit que les principaux détails de formatage du document sont décrits globalement dans ConTEXt, et que nous travaillons avec des fichiers texte qui sont –

² Au moment de la première version de ce texte, ceci était vrai ; mais au printemps 2020, le Wiki ConTeXt a été mis à jour et nous devons supposer qu'à partir de ce moment la distribution « officielle » de ConTeXt est devenue LMTX. Cependant, pour ceux qui entrent dans le monde de ConTeXt pour

la première fois, je recommande quand même d'utiliser « ConTeXt Standalone » puisque c'est une distribution plus stable. L'annexe A explique comment installer l'une ou l'autre des distributions.

³ Pour la liste, voir section 3.6.

une fois que nous nous y sommes habitués – une façon beaucoup plus naturelle de traiter la création et l'édition de documents ; d'autant plus que ces types de fichiers sont beaucoup plus légers et plus faciles à traiter que les lourds fichiers binaires des traitements de texte.

Documentation

Il existe une documentation considérable sur ConTeXt, presque exclusivement en anglais. Par exemple, ce que l'on considère comme étant la distribution *officielle* de ConTeXt – appelée « ConTeXt Standalone »² – contient une documentation de quelques 180 fichiers PDF (la majorité en anglais, mais d'autres en néerlandais et en allemand) avec notamment des manuels, des exemples et des articles techniques ; et sur le site web [Pragma ADE](#) (la société qui a donné naissance à ConTeXt) il y a (le jour où j'ai fait le décompte en mai 2020) 224 documents librement téléchargeables, dont la plupart sont distribués avec la « ConTeXt Standalone » mais également quelques autres. Cependant, cette énorme documentation n'est pas particulièrement utile durant la phase d'apprentissage de ConTeXt, car, en général, ces documents ne s'adressent pas à un lecteur désireux d'apprendre mais novice, qui ne connaît rien du système. Sur les 56 fichiers PDF que « ConTeXt Standalone » appelle « manuels », un seul suppose que le lecteur ne connaît rien sur ConTeXt. Il s'agit du document intitulé « [ConTeXt Mark IV, an Excursion](#) » ou en français « [ConTeXt Mark IV, une escapade](#) ». Mais ce document, comme son nom l'indique, se limite à présenter le système et à expliquer comment faire certaines choses qui peuvent être faites avec ConTeXt. Ce serait une bonne introduction s'il était suivi d'un manuel de référence un peu plus structuré et systématique. Mais un tel manuel n'existe pas et l'écart entre le document « [ConTeXt Mark IV, an Excursion](#) » et le reste de la documentation est trop important.

En 2001, un [manuel de référence](#) a été rédigé ; mais malgré son titre, d'une part il n'a pas été conçu pour être un manuel complet (la tâche étant titanique), et d'autre part il était (est) destiné à la version précédente de ConTeXt (appelé Mark II) et intègre donc des éléments obsolètes, ce qui perturbe l'apprentissage.

En 2013, ce manuel [a été partiellement mis à jour](#) mais beaucoup de ses sections n'ont pas été réécrites et il contient des informations relatives à la fois à ConTeXt Mark II et ConTeXt Mark IV (la version actuelle), sans toujours préciser clairement quelles informations se rapportent à chacune des versions. C'est peut-être la raison pour laquelle ce manuel ne se trouve pas parmi les documents inclus dans « ConTeXt Standalone ». Pourtant, malgré ces défauts, et une fois lu « [ConTeXt Mark IV, an Excursion](#) », le manuel reste le meilleur document pour continuer à apprendre ConTeXt. Autres informations également toujours très utiles pour démarrer avec ConTeXt, celles contenues le sur [ConTeXt Garden wiki](#), qui, au moment où nous écrivons ces lignes, est plein remaniement et présente une structure beaucoup plus claire, bien qu'elles mélangeant également des explications qui ne fonctionnent que dans Mark II avec d'autres pour Mark IV ou pour les deux versions. Ce manque de

différenciation se retrouve également dans la liste officielle des commandes « ConTeXt Commands »³ qui comprend, pour chacune d'elles, l'ensemble des options de configuration possibles mais ne précise pas quelles commandes ne fonctionnent que dans l'une ou l'autre des versions.

Fondamentalement, cette introduction a été rédigée en s'inspirant des quatre sources d'information énumérées préalablement : « ConTeXt Mark IV, an Excursion », « ConTeXt Reference Manual 2013 », le contenu de ConTeXt Garden wiki, et la liste officielle des commandes « ConTeXt Commands », en plus, bien sûr, de mes propres essais et tribulations. Ainsi, cette introduction résulte d'un effort d'investigation, et, durant un temps, j'ai été tenté de l'appeler « Ce que je sais à propos de ConTeXt Mark IV » ou « Ce que j'ai appris sur ConTeXt Mark IV ». Finalement, aussi vraie que soit leur teneur, ces titres ont été écartés car j'ai pensé qu'ils risquaient de dissuader certains de s'investir dans ConTeXt. Il est certain, malgré que la documentation ait selon moi certaines lacunes, que ConTeXt est un outil vraiment utile et polyvalent, pour lequel l'effort d'apprentissage vaut sans aucun doute la peine. **Grâce à ConTeXt, nous pouvons manipuler et configurer des documents texte pour réaliser des choses que ceux qui ne connaissent pas le système ne peuvent tout simplement pas imaginer.**

Je ne peux pas empêcher – parce que je suis comme je suis – que mes regrets concernant les déficiences d'information ressurgissent de temps en temps dans ce document. Je ne veux pas qu'il y ait de malentendu : je suis immensément reconnaissant aux créateurs de ConTeXt d'avoir conçu un outil aussi puissant et de l'avoir mis à la disposition du public. C'est simplement que je ne peux m'empêcher de penser que cet outil serait beaucoup plus populaire si sa documentation était améliorée : il faut investir beaucoup de temps pour s'approprier ConTeXt, non pas tant en raison de sa difficulté intrinsèque (qui existe, mais qui n'est pas supérieure à celle d'autres outils spécialisés similaires, c'est même plutôt le contraire), mais en raison du manque d'informations claires, complètes et systématiques, qui diffèrent les deux versions de ConTeXt, expliquent ce qui fonctionne dans chacune d'elles et, surtout, précisent à quoi sert chaque commande, argument et option.

Il est vrai que de ce type d'information exigerait un fort investissement en temps. Mais comme de nombreuses commandes partagent des options avec des noms similaires, on pourrait peut-être rédiger une sorte de *glossaire* des options, ce qui permettrait également de détecter certaines incohérences produites lorsque deux options du même nom font des choses différentes, ou lorsque des noms d'options différents sont utilisés dans des commandes différentes pour faire la même chose.

Quant au lecteur qui s'intéresse à ConTeXtpour la première fois, que mes plaintes ne le dissuadent pas, car s'il est vrai que le manque d'informations, claires complètes et systématiques, augmente le temps nécessaire à l'apprentissage, du moins pour les sujets traités dans cette introduction, j'ai déjà investi ce temps, de sorte que le lecteur n'aura pas à le refaire. Et déjà avec ce que vous aurez l'occasion d'apprendre dans cette introduction, vous pourrez produire des documents avec de puissants utilitaires insoupçonnés.

⁴ Je n'ai pas dessiné l'image moi-même, elle provient d'internet (<https://es.dreamstime.com/>), où il est indiqué qu'elle est libre de droit.

Incertitudes

Etant donné que ce qui est expliqué dans ce document provient essentiellement de mes propres conclusions, il est probable que, bien qu'ayant personnellement vérifié une grande partie de ce qui est exposé, certaines déclarations ou opinions soient incorrectes ou non orthodoxes. Bien évidemment, j'apprécierai toute correction, toute nuance ou encore précisions que vous accepterez de me faire parvenir à

joaquin@ataz.org. Pour limiter les occasions d'erreur, j'ai essayé de ne pas aborder les sujets sur lesquels je n'ai pas trouvé d'informations ou que je n'ai pas pu (ou voulu) vérifier personnellement ; parce que parfois le résultat de mon test n'était pas concluant, d'autres fois parce que je n'ai pas toujours tout essayé : le nombre de commandes et d'options de ConTeXtest impressionnant, et si je devais tout essayer, je n'aurais jamais réussi à finaliser cette introduction. Néanmoins, à certaines occasions je n'ai pas pu éviter de faire certaines *conjectures*, c'est à dire une déclaration que je considère comme probable mais dont je ne suis pas totalement sûr. Dans ce cas, en marge du paragraphe, l'image qui peut être vue à gauche de cette ligne indiquera visuellement la présence d'une conjecture⁴. D'autres fois, je n'ai pas eu d'autre choix que d'admettre que je ne sais pas et que je n'ai même pas d'hypothèse raisonnable à ce sujet : dans ce deuxième cas, l'image utilisée est celle insérée ici en marge du paragraphe afin de représenter plus que de simples conjectures, l'ignorance. Mais n'ayant jamais été très doué pour les représentations graphiques, je ne suis pas certain que les images sélectionnées parviennent vraiment à transmettre ces nuances.

Public visé

Autre aspect, cette introduction a été écrite pour un lecteur qui ne connaît rien à TeX et ConTeXt, bien que j'espère qu'elle pourra également être utile à ceux qui s'approchent pour la première fois de TeX or L^AT_EX (le plus populaire des outils dérivés de TeX). Je suis cependant conscient qu'en essayant de satisfaire des lecteurs aussi différents, je risque de n'en satisfaire aucun. Par conséquent, en cas de doute, j'ai toujours été clair sur le fait que le principal destinataire de ce document est le nouvel arrivant dans le monde de ConTeXt, le nouveau venu dans cet écosystème fascinant.

Être un néophyte ConTeXt n'implique pas d'être également néophyte dans la manipulation des outils informatiques. Bien que cette introduction ne présuppose aucun niveau spécifique de compétence informatique chez son lecteur, elle suppose une certaine « aisance raisonnable » en informatique qui implique, par exemple, de connaître dans les grandes lignes la différence entre un éditeur de texte et un traitement de texte, de savoir comment créer, ouvrir et manipuler un fichier texte, de savoir comment installer un programme, de savoir comment ouvrir un terminal et y exécuter une commande... et un peu plus.

En lisant les parties de cette introduction déjà écrites au moment de la rédaction, je me rends compte que parfois, je m'emporte et me lance dans des problèmes informatiques qui ne sont pas nécessaires à l'apprentissage de ConTeXt et peuvent effrayer le néophyte, tandis que d'autres fois, je suis occupé à expliquer des choses assez évidentes qui peuvent ennuyer le lecteur expérimenté. Je demande votre indulgence. Rationnellement, je sais qu'il est très difficile pour un total débutant en traitement de texte informatique de connaître l'existence même de ConTeXt mais, d'un autre côté, dans mon environnement professionnel, je suis entouré de personnes qui se battent continuellement en utilisant les logiciels de traitement de texte, et elles s'en sortent raisonnablement bien, mais néanmoins, n'ayant jamais travaillé avec des fichiers texte, elles ignorent des aspects aussi élémentaires que, par exemple, ce qu'est l'encodage ou la différence entre un éditeur et un traitement de texte.

Le fait que ce manuel ait été conçu pour des personnes qui ne connaissent rien à ConTeXt ou à TeX implique que j'ai dû y inclure des informations concernant non pas à ConTeXt mais à TeX, le système de base mais j'ai compris qu'il n'était pas nécessaire de surcharger le lecteur avec des informations qui ne l'intéressent guère, par exemple de savoir si une telle commande qui fonctionne *de facto* provient de ConTeXt ou bien de TeX. Par conséquent, ce n'est qu'en certaines occasions, lorsque je l'ai considéré utile, qu'il est précisé que certaines commandes appartiennent réellement à TeX.

Structuration

En ce qui concerne l'organisation de ce document, le contenu est présenté en trois parties :

- **La première partie**, qui comprend les quatre premiers Chapitres, donne une vue d'ensemble de ConTeXt, en expliquant ce que c'est, comment l'utiliser, en montrant un premier exemple de transformation d'un document, pour ensuite expliquer certains concepts fondamentaux de ConTeXt ainsi que certaines questions relatives aux fichiers sources de ConTeXt

Dans l'ensemble ces chapitres sont destinés aux lecteurs qui ne savaient travailler jusqu'à présent qu'avec des traitements de texte. Un lecteur qui connaît déjà l'utilisation des langages de balisage peut sauter les deux premiers chapitres. Si le lecteur connaît déjà TeXou L^ATeX, il peut également sauter une grande partie du contenu des Chapitres 3 et 4. Mais je vous recommande quand même de lire au moins :

- les informations relatives aux commandes de ConTeXt ([Chapitre 3](#)), et en particulier sur la configuration de son fonctionnement, car c'est là que réside la principale différence dans les conceptions et syntaxes de L^ATeX et ConTeXt. Comme cette introduction ne concerne que ce dernier, ces différences ne sont pas expressément mentionnées comme telles, mais quiconque lit ce chapitre et connaît le fonctionnement de L^ATeX comprendra immédiatement les principales différences dans la syntaxe des deux langages, ainsi que la façon dont ConTeXt permet de configurer et de personnaliser le fonctionnement de presque toutes ses commandes.
- Les informations relatives à la gestion des projets ConTeXt multi-fichiers ([Chapitre 4](#)), qui se distingue de celles des autres systèmes basés sur TeX.

- **La seconde partie**, qui comprend les Chapitres 5 à 9, se concentre sur ce que nous pouvons considérer comme l'aspect général d'un document :

- Les deux aspects qui affectent principalement l'apparence d'un document sont les dimensions et la composition de ses pages ainsi que la police utilisée. Les chapitres [5](#) et [6](#) sont consacrés à ces deux questions.
 - * Le [Chapitre 5](#) se concentre sur les pages : taille, éléments qui la composent, conception ou composition (c'est-à-dire répartition des différents éléments sur la page), etc. Pour une raison de systématisme, des aspects plus spécifiques sont également traités ici, comme ceux relatifs à la pagination et aux mécanismes qui nous permettent de l'influencer.

- * Le [Chapitre 6](#) explique les commandes relatives aux polices et à leur manipulation. Une explication de base de l'utilisation et de la manipulation des couleurs est également incluse ici, car, bien que les couleurs ne soient pas, à proprement parler, une *caractéristique* de la police, elles influencent de la même manière l'apparence visuelle du document.
 - Les chapitres [7](#) et [8](#) se concentrent sur la structure du document et les outils que ConTeXt met à la disposition de l'auteur pour la rédaction de documents bien structurés. Le [Chapitre 7](#) se concentre sur la structure elle-même (divisions structurelles du document) et le [Chapitre 8](#) sur la valorisation de cette structure dans une table des matières.
 - Enfin, le [Chapitre 9](#) se concentre sur l'aspect clé de l'utilisation de références par un document, références vers d'autres points du même document (références internes) mais aussi vers des documents externes (références externes). Dans cette dernière situation, nous n'aborderons que le cas de références impliquant un *liens* vers le document externe. Ce chapitre explique l'utilisation de l'outil de ConTeXtpour la gestion de ces références qui permet des *sauts* entre différents zones d'informations, internes ou externes, et rend ainsi notre document *interactif*.
- Ces chapitres n'ont pas besoin d'être lus dans un ordre particulier, sauf peut-être le [Chapitre 8](#), qui est peut-être plus facile à comprendre si vous avez d'abord lu le [Chapitre 7](#). En tout cas, j'ai essayé de faire en sorte que lorsqu'une question se pose dans un chapitre ou une section, alors qu'elle est traitée ailleurs dans ce document, le texte mentionne ce fait et propose un hyperlien vers le point où cette question est traitée. Je ne suis cependant pas en mesure de garantir que ce sera toujours le cas.
- Enfin, **la troisième partie** ([Chapitre 10](#) et suivants) se concentre sur des aspects plus concrets, plus spécialisés. Non seulement les chapitres sont maintenant indépendants les uns des autres, mais également les sections les unes des autres au sein d'un même chapitre (sauf, peut-être, dans le dernier chapitre). Étant donné la grande quantité de fonctionnalités proposées par ConTeXt cette troisième partie pourrait être très vaste ; mais comme je devine qu'en arrivant à ce stade le lecteur sera capable de plonger lui-même dans la documentation de ConTeXt je n'ai considéré que les chapitres suivants :
 - Les chapitres [10](#) et [11](#) traitent de ce que l'on pourrait appeler les *éléments clés* de tout document texte : Le texte est constitué de caractères, qui forment des mots, qui sont regroupés en lignes, qui à leur tour forment des paragraphes, qui sont séparés les uns des autres par un espace vertical... Il est évident que toutes ces questions auraient pu être incluses dans un seul chapitre. Mais comme cela aurait été trop long, j'ai divisé cette question en deux chapitres, l'un traitant des caractères, des mots et des espaces horizontaux, et l'autre traitant des lignes, des paragraphes et des espaces verticaux.
 - Le [Chapitre 12](#) est une sorte de *pot-pourri* qui traite des éléments et constructions qui sont communs à de nombreux documents, principalement s'ils sont académiques ou scientifico-techniques : notes de bas de page, listes structurées, descriptions, énumérations, etc.
 - Enfin, le [Chapitre 13](#) se concentre sur les objets flottants insérés dans un document et en particulier les deux cas les plus répandus : les images et les tableaux.

- Cette introduction à ConTeXt se termine par trois **Annexes**. I L'une concerne l'installation de ConTeXt sur un ordinateur, une deuxième annexe présentent plusieurs dizaines de commandes qui permettent de générer divers symboles (principalement, mais pas uniquement, pour un usage mathématique), et une troisième annexe rassemble les commandes ConTeXt expliquées ou mentionnées tout au long de ce texte sous la forme d'une liste alphabétique.

Reste à faire

De nombreux points restent à expliquer : le traitement des citations et des références bibliographiques, la rédaction de textes spéciaux (mathématiques, chimie...), la connexion avec XML, l'interface pour le code Lua, les modes et la compilation basée sur les modes, la collaboration avec MetaPost pour le design graphique, etc. Par conséquent, comme ce document ne contient pas d'explication complète de ConTeXt et ne prétend pas le faire, j'ai intitulé ce document « une courte (?) introduction à ConTeXt Mark IV » , et ajouté entre parenthèses l'observation « pas trop courte » car, de toute évidence, elle l'est. Un texte qui laisse tant de choses dans l'encrier et qui dépasse pourtant 300 pages n'est certainement pas une « courte introduction ». C'est parce que j'essaie de faire comprendre au lecteur la logique de ConTeXt ; ou du moins la logique telle que je la comprends. Il n'est pas destiné à être un manuel de référence, mais plutôt un guide d'auto-apprentissage qui prépare le lecteur à réaliser des documents de complexité moyenne (ce qui inclut la plupart des documents possibles) et qui, surtout, lui apprend à *enviser et imaginer* ce qui peut être fait avec ce puissant outil et à *repérer* dans la documentation comment le faire. Ce document n'est pas non plus un tutoriel. Les tutoriels sont conçus pour augmenter progressivement la difficulté, afin que ce qui doit être enseigné soit appris pas à pas ; j'ai, en ce sens, préféré, dès la deuxième partie, être plus systématique au lieu d'ordonner le sujet en fonction de sa difficulté. Mais, même si ce n'est pas un tutoriel, j'ai inclus de nombreux exemples.

Il est possible que le titre de ce document rappelle à certains lecteurs celui écrit par OETIKER, PARTL, HYNA ET SCHLEGL, en anglais « *The Not So Short Introduction to L^AT_EX 2_e* » et en français « *Une courte (?)introduction à L^AT_EX 2_e* ». Ce document, [disponible en 24 langues](#), est l'un des meilleurs documents pour se familiariser avec le monde de L^AT_EX. Ce n'est pas une coïncidence, mais un hommage et un acte de gratitude : grâce au travail généreux de ceux qui écrivent de tels textes, il est possible pour de nombreuses personnes de s'initier à des outils utiles et puissants comme L^AT_EX et ConTeXt. Ces auteurs m'ont aidé à me lancer dans L^AT_EX ; j'ai l'intention de faire de même pour ceux qui veulent se lancer dans le ConTeXt, bien que je sois limité au public hispanophone, qui, par ailleurs, manque tellement de documentation dans sa langue. J'espère que ce document répondra à cet objectif.

Joaquín Ataz-López
Eté 2020

I

Qu'est ce que ConTEXt et comment travailler avec ?

Chapitre 1

ConTeXt : une vue d'ensemble

Table of Contents : 1.1 Qu'est-ce que ConTeXt ? ; 1.2 La composition typographique de document ; 1.3 Les langages de balisage ; 1.4 TeX et ses dérivés ; 1.4.1 Les moteurs TeX ; 1.4.2 Les formats dérivés de TeX ; 1.5 ConTeXt ; 1.5.1 Une brève histoire de ConTeXt ; LMTX et autres implémentations alternatives de Mark IV 30 1.5.2 ConTeXt versus LATEX ; 1.5.3 La logique de travail avec ConTeXt ; 1.5.4 Obtenir de l'aide sur ConTeXt ;

1.1 Qu'est-ce que ConTeXt ?

ConTeXt est un *système de composition*, c'est à dire un ensemble complet d'outils visant à donner à l'utilisateur un contrôle le plus complet précis possible sur la présentation et l'apparence d'un document électronique destiné à être imprimé sur papier ou affiché à l'écran. Ce chapitre expliquera ce que cela signifie. Mais d'abord, mettons en évidence certains des caractéristiques de ConTeXt.

- Il existe deux versions de ConTeXt appelées respectivement Mark II et Mark IV. ConTeXt Mark II est *gelé*, c'est-à-dire qu'il est considéré comme finalisé, qu'aucun changement ou nouvelle fonctionnalité ne devrait y être introduit. Une nouvelle version ne sera publiée que si un bogue est détecté et doit être corrigé. ConTeXt Mark IV, en revanche, est toujours en développement, de nouvelles versions intègrent périodiquement des améliorations et fonctionnalités supplémentaires. Mais, bien que toujours en cours de développement, c'est un langage très mature, dans lequel les changements introduits par les nouvelles versions sont très subtils et n'affectent que le fonctionnement de bas niveau du système. Pour l'utilisateur moyen ces changements sont totalement transparents, c'est comme s'ils n'existaient pas. Pour finir, bien que les deux versions aient beaucoup en commun, il y a aussi quelques incompatibilités entre elles. Cette introduction se concentre donc uniquement sur le ConTeXt Mark IV.
- ConTeXt est un logiciel libre. Le programme lui-même (c'est-à-dire l'ensemble des outils logiciels qui composent ConTeXt) est distribué sous la *Licence Publique Générale GNU*. La documentation est fournie sous une licence *Creative Commons* qui vous permet de la copier et de la distribuer librement.

⁵ ConTeXt désigne donc à la fois un langage et un programme (ainsi qu'un ensemble d'autres outils formant le système complet).

Par conséquent, dans un texte comme cette introduction, se pose le problème de devoir parfois faire la distinction entre les deux aspects. J'ai donc adopté la convention typographique consistant à écrire «ConTeXt» avec son logo (ConTeXt) lorsque je veux me référer exclusivement à la langue, ou indistinctement à la langue et au programme. Et lorsque je veux me référer exclusivement au programme j'écrirai « context » tout en minuscules et avec une police de caractères à es-

pace-ment constant, typique des terminaux d'ordinateur et des machines à écrire, que j'utiliserais aussi pour les exemples et pour faire référence aux instructions du langage.

- ConTeXt n'est pas un programme de traitement de texte ou d'édition de texte, mais un ensemble d'outils conçus pour *transformer* un texte que nous avons précédemment écrit avec notre éditeur de texte préféré. Par conséquent, lorsque nous travaillons avec ConTeXt :
 - Nous commençons par écrire un ou plusieurs fichiers texte avec n'importe quel éditeur de texte.
 - Dans ces fichiers, en plus du texte qui constitue le contenu réel du document, il y a une série d'instructions, instructions qui indiquent à ConTeXt à quoi doit ressembler le document final généré à partir des fichiers texte originaux. L'ensemble complet des instructions ConTeXt constitue en fait un *langage* ; et puisque ce langage permet de *programmer* la transformation typographique d'un texte, on peut dire que ConTeXt est un *langage de programmation typographique*.
 - Une fois les fichiers sources écrits, ils seront traités par un programme (également appelé « *context* »⁵), qui, à partir de ceux-ci, générera un fichier PDF prêt à être envoyé à une imprimante ou à être affiché à l'écran.
- Dans ConTeXt, nous devons donc faire la différence entre le document que nous rédigons et le document que ConTeXt génère. Pour éviter tout doute, dans cette introduction, j'appellerai *fichier source* le document texte qui contient les instructions de formatage, et *document final* le fichier PDF généré par ConTeXt à partir du fichier source.

Dans la suite, les points fondamentaux ci-dessus seront développés un peu plus.

1.2 La composition typographique de document

Ecrire un document (livre, article, chapitre, brochure, dépliant, imprimé, affiche...) et le mettre en page, dit également le composer, sont deux activités très différentes.

L'écriture du document c'est sa rédaction, qui est effectuée par l'auteur, qui décide de son contenu et de sa structure. Le document créé directement par l'auteur, tel qu'il l'a écrit, s'appelle un *manuscrit*. Le manuscrit, par sa nature même, n'est accessible qu'à l'auteur et aux personnes à qui l'auteur permet de le lire. Sa diffusion au-delà de ce cercle intime nécessite que le manuscrit soit *publié*. De nos jours, publier quelque chose — au sens étymologique de « rendre accessible au public » — est aussi simple que de le mettre sur l'internet, à la disposition de quiconque peut le localiser et souhaite le lire. Mais jusqu'à une date relativement récente, l'édition était un processus qui impliquait des coûts, dépendait de certains professionnels spécialisés dans ce domaine et n'était accessible qu'aux manuscrits qui, en raison de leur contenu ou de leur auteur, étaient considérés comme particulièrement intéressants. Et aujourd'hui encore, nous avons tendance à réservier le mot *publication* au type de *publication professionnelle* par lequel le manuscrit subit une série de transformations de son apparence visant à améliorer la *lisibilité du document*. C'est cette série de transformations qui est appelée *composition* ou encore *composition typographique*.

L'objectif de la composition est — en général, et en laissant de côté les textes publicitaires qui cherchent à attirer l'attention du lecteur — de réaliser des documents avec une *lisibilité maximale*, entendue comme la qualité d'un texte imprimé qui invite à la lecture, ou qui la facilite, et qui fait que le lecteur s'y sente à l'aise. De nombreux aspects y contribuent ; certains, bien sûr, sont liés au *contenu* du document (qualité, clarté, standardisation...), mais d'autres dépendent de questions telles que le type et la taille de la police utilisée, la répartition des espaces blancs sur la page, la séparation visuelle entre les paragraphes, etc., ou encore d'autres outils, moins graphiques ou visuels, telles que l'existence ou non dans le document de certaines aides à la lecture comme les en-têtes ou les pieds de page, les index, les glossaires, les caractères gras, les titres dans les marges, etc. On pourrait appeler « art de la composition » ou « art de l'impression » la connaissance et la manipulation correcte de toutes les ressources dont dispose un compositeur, un imprimeur.

Historiquement, et jusqu'à l'avènement des ordinateurs, les tâches et les rôles du rédacteur et du compositeur sont restés clairement différenciés. L'auteur écrivait à la main ou, depuis le milieu du XIX^e siècle, sur une « machine à écrire » dont les ressources typographiques étaient encore plus limitées que celles de l'écriture manuscrite ; puis il remettait ses originaux à l'éditeur ou à l'imprimeur qui se chargeait de les transformer pour en tirer le document imprimé.

De nos jours, grâce à l'informatique, il est plus facile pour l'auteur lui-même de définir la composition jusqu'aux moindres détails. Mais pour autant les qualités d'un bon auteur ne sont pas les mêmes que celles d'un bon compositeur. L'auteur doit avoir une bonne connaissance du sujet traité, savoir le structurer, l'exposer avec clar-

⁶ Par une convention assez ancienne, on fait une distinction entre les logiciels d'édition de texte et les logiciels de *traitement de texte*. Les premiers manipulent des fichiers de texte brut, et les seconds des fichiers de texte au format binaire permettant une plus grande complexité.

té, avec créativité, avec un rythme. etc. Le compositeur typographe doit avoir une bonne connaissance de l'environnement graphique et conceptuel à sa disposition, un goût de l'esthétique pour les utiliser harmonieusement, de façon cohérente avec le sujet traité, avec les tendances du moment.

Avec un bon logiciel de traitement de texte⁶, il est possible d'obtenir une composition raisonnablement bonne. Mais les traitements de texte ne sont généralement pas conçus pour la composition et leurs résultats, même s'ils sont corrects, ne sont pas comparables à ceux obtenus avec d'autres outils spécifiquement conçus pour contrôler la composition des documents. En fait, les traitements de texte sont l'évolution des machines à écrire, et leur utilisation, dans la mesure où ces outils masquent la différence entre la rédaction du texte (la paternité) et sa composition, tend à produire des textes parfois moins structurés et moins bien optimisés typographiquement.

Au contraire, les outils tels que ConTEXt sont l'évolution de l'imprimerie ; ils offrent beaucoup plus de possibilités de composition et, surtout, il n'est pas possible d'apprendre à les utiliser sans acquérir également, en cours de route, de nombreuses notions liées à la composition, contrairement aux traitements de texte, qui peuvent être utilisés pendant de nombreuses années sans apprendre un seul mot de typographie.

1.3 Les langages de balisage

Avant l'arrivée de l'informatique, comme je l'ai déjà dit, l'auteur préparait son manuscrit à la main ou à la machine à écrire et le remettait à l'éditeur ou à l'imprimeur, qui était chargé de transformer le manuscrit en texte final imprimé. Bien que l'auteur soit relativement peu intervenu dans cette transformation, il l'a fait dans une certaine mesure, par exemple en indiquant que certaines lignes du manuscrit étaient les titres de ses différentes parties (chapitres, sections...) ; ou en indiquant que certains fragments devaient être mis en valeur typographiquement d'une certaine manière. Ces indications étaient faites par l'auteur dans le manuscrit lui-même, parfois expressément, et d'autres fois au moyen de certaines conventions qui, avec le temps, se sont développées ; ainsi, par exemple, les chapitres commençaient toujours sur une nouvelle page, en insérant plusieurs lignes vierges avant le titre, en le soulignant, en l'écrivant en majuscules ; ou en encadrant le texte à mettre en valeur entre deux soulignements, en augmentant l'indentation d'un paragraphe, etc.

L'auteur, en somme, *indiquait* dans le texte original quelques éléments concernant la composition typographique du texte. L'éditeur ensuite inscrivait à son tour de nouvelles indication pour l'imprimeur, comme par exemple la police et la taille de caractères.

Aujourd'hui, dans un monde informatisé, nous pouvons continuer à faire de même pour la génération de documents électroniques, au moyen de ce que l'on appelle un *langage de balisage*. Dans ce type de langage, on utilise une série de marques ou d'indications ou encore de *balises* que le programme traitant le fichier qui les contient sait interpréter. Le langage de balisage le plus connu au monde aujourd'hui est sans doute le HTML, car la plupart des pages web sont basées sur ce langage. Un fichier HTML contient le texte d'une page web, ainsi qu'une série de marques qui indiquent au programme de navigation avec lequel la page est chargée, comment elle doit être affichée. L'ensemble des balises HTML compréhensibles par les navigateurs web, ainsi que les instructions sur la manière et l'endroit où les utiliser, est appelé « langage HTML », qui c'est un langage de balisage. Mais en plus du HTML, il existe de nombreux autres langages de balisage ; en fait, ceux-ci sont en plein essor et ainsi, le XML, qui est le langage de balisage par excellence, est aujourd'hui absolument omniprésent et est utilisé pour presque tout : pour la conception de bases de données, pour la création de langages spécifiques, pour la transmission de données structurées, pour les fichiers de configuration d'applications, et ainsi de suite. Il existe également des langages de balisage conçus pour la conception graphique (SVG, TikZ ou MetaPost), les formules mathématiques (MathML), la musique (Lilypond et MusicXML), la finance, la géographie, etc. Il y a aussi, bien sûr, ceux destinés à la transformation typographique des textes, et parmi eux se distinguent TeX et ses dérivés.

En ce qui concerne les balises *typographiques*, qui indiquent l'apparence que doit avoir un texte, il en existe deux types, que nous pourrions distinguer comme d'un côté les *balises purement typographique* (ou encore graphiques) et de l'autre les *balises sémantiques* (ou encore conceptuelles, logiques). Les balises purement typographiques se limitent à indiquer précisément quelles ressources typographiques

doivent être utilisées pour afficher un certain texte ; par exemple, lorsque nous indiquons qu'un certain texte doit être en gras ou en italique, de telle ou telle couleur. Le balisage sémantique, quant à lui, indique la fonction d'un texte donné dans l'ensemble du document, par exemple lorsque nous indiquons qu'il s'agit d'un titre, d'un sous-titre, d'une citation. En général, les documents qui utilisent de préférence ce deuxième type de balisage sont plus cohérents et plus faciles à composer, car la différence entre la paternité et la composition y est à nouveau claire : l'auteur indique que cette ligne est un titre, ou que ce fragment est un avertissement, ou une citation ; et le compositeur décide comment mettre en valeur typographiquement tous les titres, avertissements ou citations ; ainsi, d'une part, la cohérence est garantie, puisque tous les fragments remplissant la même fonction auront la même apparence, et, d'autre part, on gagne du temps, puisque le format de chaque type de fragment ne doit être indiqué qu'une seule fois.

1.4 TeX et ses dérivés

TeX a été développé à la fin des années 1970 par DONALD E. KNUTH, professeur de théorie de la programmation à l'université de Stanford, qui l'a utilisé pour composer ses propres publications et ainsi que pour donner un exemple de *programmation littéraire*, une approche de la programmation où le code source du logiciel est systématique commenté et documenté. Avec TeX, Knuth a également développé un langage de programmation supplémentaire appelé METAFONT, pour la conception de caractères typographiques, avec lequel il a créé une police qu'il a nommée *Computer Modern*, qui, en plus des caractères habituels de toute police, comprenait également un ensemble complet de « glyphs »⁷ pour l'écriture des mathématiques. Il a ajouté à tout cela quelques utilitaires supplémentaires et c'est ainsi qu'est né le système de composition appelé TeX, qui, pour sa puissance, la qualité de ses résultats, sa flexibilité d'utilisation et ses vastes possibilités, est considéré comme l'un des meilleurs systèmes informatiques pour la composition de textes. Il a été pensé pour des textes dans lesquels il y avait beaucoup de mathématiques, mais on a vite vu que les possibilités du système le rendaient adapté à tous les types de textes.

En interne, il fonctionne comme la machine à écrire d'une presse à imprimer, car tout y est *boîte*. Les lettres sont contenues dans des boîtes, les blancs sont aussi des boîtes. Un mot est une boîte enfermant les boîtes de ses lettres. Une ligne est une boîte enfermant les boîtes de ses mots et des blancs entre ces mots. Un paragraphe est une boîte contenant l'ensemble des boîtes de ses lignes. Et ainsi de suite. Tout cela avec une précision extraordinaire apportée au traitement des mesures. Il suffit de penser que la plus petite unité que TeX traite est 65,536 fois plus petite que le point typographique, avec lequel on mesure les caractères et les lignes, qui est généralement la plus petite unité traitée par la plupart des programmes de traitement de texte. Cette plus petite unité traité par TeX est d'environ 0,000005356 millimètre.

Le nom TeX vient de la racine du mot grec τέχνη, écrit en lettres capitales (TÉXNH). Par conséquent, comme la dernière lettre du nom n'est pas un « X » latin, mais le « χ » grec, il faut prononcer « Tec ». Ce mot grec, quant à lui, signifiait à la fois « art » et « technique », c'est pourquoi Knuth l'a choisi comme nom pour son système. Le but de ce nom, écrit-il, « est de rappeler qu'il s'occupe principalement de manuscrits techniques de haute qualité. Elle met l'accent sur l'art et la technologie, tout comme le mot grec sous-jacent ». Par convention établie par Knuth, le nom de est à écrire :

- Dans des textes formatés typographiquement, comme le présent texte, en utilisant le logo que j'ai utilisé jusqu'à présent : Les trois lettres sont en majuscules, avec le « E » central légèrement décalé vers le bas pour faciliter un rapprochement entre le « T » et le « X » ; c'est-à-dire : « TeX ». Pour rendre plus facile l'écriture d'un tel logo, Knuth a inclus dans une instruction qui l'inscrit dans le document final : TeXTeX.
Pour rendre plus facile l'écriture d'un tel logo, Knuth a inclus dans une instruction qui l'inscrit dans le document final : \TeX.
- Dans un texte non formaté (tel qu'un e-mail ou un fichier texte), le « T » et le « X » sont en majuscules, et le « e » du milieu est en minuscules ; par exemple : « TeX ».

⁷ En typographie, un glyphe est une représentation graphique d'un caractère, de plusieurs caractères ou d'une partie d'un caractère et est l'équivalent actuel du type d'impression (la pièce mobile en bois ou en plomb qui portait la gravure de la lettre).

Cette convention est suivie dans tous les dérivés de \TeX qui l'incluent dans leur propre nom, comme par exemple Con \TeX t, qui lorsqu'il est écrit en mode texte doit être écrit « Con \TeX t ».

1.4.1 Les moteurs \TeX

Le programme \TeX est un logiciel libre : son code source est à la disposition du public et chacun peut l'utiliser ou le modifier à sa guise, à la seule condition que, si des modifications sont introduites, le résultat ne puisse être appelé « \TeX ». C'est la raison pour laquelle, au fil du temps, certaines adaptations du programme sont apparues, qui lui ont apporté différentes améliorations, et qui sont généralement appelées *moteurs \TeX* (engine en anglais). En dehors du programme original, les principaux moteurs \TeX sont, par ordre chronologique d'apparition pdf \TeX , ε - \TeX , $\text{X}\varepsilon\text{\TeX}$ et Lua \TeX . Chacun d'entre eux est censé intégrer les améliorations de son pré-décesseur. Ces améliorations, en revanche, jusqu'à l'apparition de Lua \TeX , n'ont pas affecté le langage lui-même, mais seulement les fichiers d'entrée, les fichiers de sortie, la gestion des polices et le fonctionnement de bas niveau des macros.

La question du choix du moteur \TeX à utiliser fait l'objet d'un débat animé dans l'univers \TeX . Je ne m'y attarderai pas ici, car Con \TeX t Mark IV ne fonctionne qu'avec Lua \TeX . En fait, dans le monde de Con \TeX t la discussion sur les moteurs devient une discussion sur l'utilisation de Mark II (qui fonctionne avec pdf \TeX et $\text{X}\varepsilon\text{\TeX}$) ou Mark IV (qui fonctionne avec Lua \TeX).

1.4.2 Les formats dérivés de \TeX

Le noyau, ou cœur, de \TeX contient seulement un ensemble d'environ 300 instructions, appelées *primitives*, qui conviennent aux opérations de composition et aux fonctions de programmation très basiques. Ces instructions sont pour la plupart de très *bas niveau*, ce qui, en terminologie informatique, signifie qu'elles se rapprochent des opérations élémentaires de l'ordinateur, dans un langage machine peu approprié aux êtres humains, du type « déplacer ce caractère de 0,000725 millimètre vers le haut ».

Pour cette raison, KNUTH a rendu \TeX extensible, c'est-à-dire disposant d'un mécanisme permettant de définir des instructions de plus haut niveau, dans un langage plus facilement compréhensibles par les êtres humains. Ces instructions, qui au moment de l'exécution sont décomposées en instructions élémentaires, sont appelées *macros*. Par exemple, l'instruction \TeX qui imprime votre logo ($\backslash\text{\TeX}$), est décomposée lors de son exécution en :

```
T
\kern -.1667em
\lower .5ex
\hbox {E}
\kern -.125em
X
```

On comprend là qu'il est beaucoup plus facile pour un être humain de comprendre et mémoriser la simple commande « $\backslash\text{\TeX}$ » dont l'exécution effectue l'ensemble des opérations typographiques nécessaires à l'impression du logo.

La différence entre les *macros* et les *primitives* n'est vraiment importante que du point de vue du développeur de \TeX . Du point de vue de l'utilisateur, ce sont toutes des *instructions* ou, si vous préférez, des *commandes*. Knuth les appelait des *séquences de contrôle*.

Cette possibilité d'étendre le langage par le biais de *macros* est l'une des caractéristiques qui ont fait de \TeX un outil si puissant. En fait, KNUTH lui-même a conçu environ 600 macros qui, avec les 300 primitives, constituent le format appelé « Plain \TeX ». Il est assez courant de confondre \TeX lui-même avec Plain \TeX et, en fait, presque tout ce qui est dit ou écrit sur \TeX , se réfère en fait à Plain \TeX . Les livres qui prétendent être sur \TeX (y compris le livre fondateur « *The \TeXBook* »), font en fait référence à Plain \TeX ; et ceux qui pensent qu'ils manipulent directement \TeX manipulent en fait Plain \TeX .

Plain \TeX est ce que l'on appelle dans la terminologie \TeX un *format*, constitué d'un vaste ensemble de macros, ainsi que de certaines règles syntaxiques sur la manière et la façon de les utiliser. En plus de Plain \TeX , d'autres formats ont été développés au fil du temps, notamment L^AT_EX un vaste ensemble de macros pour \TeX conçu en 1985 par LESLIE LAMPORT, qui est probablement le dérivé de \TeX le plus utilisé dans le monde universitaire, technologique et mathématique. ConTeXt est (ou a commencé à être), de même que L^AT_EX un format dérivé de \TeX .

Normalement, ces *formats* sont accompagnés d'un programme qui charge dans la mémoire de l'ordinateur les macros qui les composent avant d'appeler « `tex` » (ou l'un des autres moteurs précédemment listés) pour traiter le fichier source. Mais bien que tous ces formats exécutent finalement \TeX , comme chacun possède ses instructions et ses règles syntaxiques spécifiques, du point de vue de l'utilisateur, nous pouvons les considérer comme des *langages différents*. Ils sont tous inspirés de \TeX , mais différents de \TeX , et différents les uns des autres.

1.5 ConTeXt

En fait, si ConTeXt a commencé comme un *format* de TeX, aujourd’hui il est beaucoup plus que cela. ConTeXt comprend :

1. Un très large ensemble de macros de TeX. Si Plain TeX se compose d’environ 900 instructions, il en compte près de 3500 ; et si l’on ajoute les noms des différentes options que ces commandes prennent en charge, on parle d’un vocabulaire d’environ 4000 mots. Le vocabulaire est aussi large car la stratégie de ConTeXtpour faciliter son apprentissage est d’inclure de nombreux synonymes des commandes et des options.
Ce qui est prévu, pour obtenir un certain effet, c’est de fournir à l’utilisateur l’ensemble des façons dont celui ci pourrait chercher à appeler cet effet. Par exemple, pour obtenir simultanément un caractère gras (en anglais *bold*) et italique (en anglais *italic*), ConTeXtpropose trois instructions identiques en terme de résultat : `\bi`, `\italicbold` y `\bolditalic`.
2. Un autre ensemble assez complet de macros pour MetaPost, un langage de programmation graphique dérivé de METAFONT, qui, lui-même, est le langage de conception de caractères que KNUTH a co-développé avec TeX.
3. Plusieurs *scripts* développés en PERL (les plus anciens), RUBY (certains également anciens et d’autres moins) et LUA (les plus récents).
4. Une interface qui intègre TeX, MetaPost, LUA et XML, permettant d’écrire et de traiter des documents dans n’importe lequel de ces langages, ou qui mélangeant des éléments de certains d’entre eux.

Vous n’avez pas compris grand-chose à l’explication ci-dessus ? Ne vous inquiétez pas. J’ai utilisé beaucoup de jargon informatique et mentionné beaucoup de programmes et de langages. Mais il n’est pas nécessaire de savoir d’où viennent les différents composants pour les utiliser. L’important, à ce stade de l’apprentissage, est de garder à l’esprit qu’il intègre de nombreux outils d’origines diverses qui forment un *système de composition typographique*.

C’est en raison de cette intégration d’outils d’origines différentes que l’on caractérise ConTeXt de « technologie hybride » dédié à la composition typographique de documents. C’est également ce qui fait de ConTeXt un système extraordinairement avancé et puissant.

Mais bien que ConTeXt soit bien plus qu’un ensemble de macros pour TeX, ses fondamentaux restent basés sur TeX, et donc ce document, qui se veut n’être qu’une *introduction*, se concentre sur cet aspect.

ConTeXt en revanche est beaucoup plus moderne que TeX. Lorsque TeX a été conçu, l'informatique commençait à peine à émerger, et on était encore loin d'entrevoir ce que serait (ce qui allait devenir) l'Internet et le monde du multimédia. En ce sens, ConTeXt intègre naturellement certains éléments qui ont toujours constitué une sorte de corps étranger, tels que l'inclusion de graphiques externes, le traitement des couleurs, les hyperliens dans les documents électroniques, l'hypothèse d'un format de papier adapté d'un affichage sur écran, etc.

1.5.1 Une brève histoire de ConTeXt

ConTeXt est né vers 1991. Il a été créé par HANS HAGEN et TON OTTEN au sein d'une société néerlandaise de conception et de composition de documents appelée « *Pragma Advanced Document Engineering* », souvent abrégée en Pragma ADE. Il s'agissait au départ d'un ensemble de macros TeX en néerlandais, officieusement connu sous le nom de *Pragmatex*, et destiné aux employés non techniques de l'entreprise, qui devaient gérer les nombreux détails de la mise en page des documents à éditer, et qui n'étaient pas habitués à utiliser des langages de balisage et des interfaces dans une autre langue que le néerlandais.

La première version de ConTeXt a donc été écrite en néerlandais. L'idée était de créer un nombre suffisant de macros avec une interface uniforme et cohérente. Vers 1994, le *paquet* était suffisamment stable pour qu'un manuel d'utilisation soit écrit en néerlandais, et en 1996, à l'initiative de HANS HAGEN, le nom « ConTeXt » a été utilisé pour s'y référer. Ce nom est censé signifier « Texte avec TeX » (en utilisant la préposition latine "con" qui a la même signification qu'en espagnol), mais il joue en même temps avec le terme « Contexte », qui en néerlandais (comme en anglais) s'écrit « context ». Derrière ce nom, il y a donc un triple jeu de mots entre « TeX », « texte » et « contexte ».

Par conséquent, bien que ConTeXt soit dérivé de TeX (prononcé « Tec »), il ne devrait pas être prononcé « *Context* » afin de ne pas perdre ce jeu de mots.

L'interface a commencé à être traduite en anglais vers 2005, donnant lieu à la version connue sous le nom de ConTeXt Mark II, où le « II » s'explique par le fait que dans l'esprit des développeurs, la version « I » est la version précédente en néerlandais, même si elle n'a jamais vraiment été appelée ainsi. Après la traduction de l'interface en anglais, le système a commencé à être utilisé en dehors des Pays-Bas, et l'interface a été traduite dans d'autres langues européennes comme le français, l'allemand, l'italien et le roumain. La documentation officielle de ConTeXt est généralement écrite sur la version anglaise, et c'est donc sur cette version que nous travaillons dans ce document, même si l'auteur de ce document (c'est-à-dire moi) est plus à l'aise en espagnol qu'en anglais.

Dans sa version initiale, ConTeXt Mark II fonctionnait avec le moteur TeX pdfTeX. Plus tard, lorsque le nouveau moteur XeTeX est apparu, ConTeXt Mark II a été modifié pour en permettre l'utilisation, qui présentait de nombreux avantages par rapport à pdfTeX. Des années plus tard encore, lorsque le moteur LuaTeX a été développé, il a été décidé de reconfigurer le fonctionnement interne de ConTeXt Mark II pour intégrer toutes les nouvelles possibilités offertes par ce dernier moteur. C'est

ainsi qu'est né ConTEXt Mark IV, qui a été présenté en 2007, immédiatement après la présentation de LuaTeX. La décision d'adapter ConTEXt à LuaTeX a très probablement été influencée par le fait que deux des trois principaux développeurs de ConTEXt, HANS HAGEN et TACO HOEKWATER, font également partie de l'équipe de développement de LuaTeX. Par conséquent, ConTEXt Mark IV et LuaTeX sont nés simultanément et ont été développés à l'unisson. Il existe une synergie entre ConTEXt et LuaTeX et qui n'existe avec aucun autre dérivé de TeX ; et je ne pense pas qu'aucun d'entre eux ne profite des possibilités de LuaTeX comme ConTEXt le fait.

Entre Mark II et Mark IV, il existe de nombreuses différences, bien que la plupart d'entre elles soient *internes*, c'est-à-dire qu'elles concernent le fonctionnement de la macro à un bas niveau, de sorte que du point de vue de l'utilisateur, la différence n'est pas perceptible : le nom et les paramètres de la macro sont les mêmes. Il existe cependant quelques différences qui affectent l'interface et vous obligent à faire les choses différemment selon la version avec laquelle vous travaillez. Ces différences sont relativement peu nombreuses, mais elles affectent des aspects très importants comme, par exemple, l'encodage du fichier d'entrée, ou la gestion des polices installées dans le système.



Cependant, il serait apprécié qu'il existe quelque part un document expliquant (ou listant) les différences significatives entre Mark II et Mark IV. Dans le wiki de ConTEXt, par exemple, il existe parfois *deux syntaxes* (souvent identiques) pour chaque commande. Je suppose que l'une est la version Mark II et l'autre la version Mark IV ; et à deviner, je suppose également que la première version est la version Mark II. Mais en pratique rien n'est indiqué à ce sujet sur le wiki.

Le fait que, pour les utilisateurs, les différences au niveau du langage soient relativement peu nombreuses, signifie que dans de nombreux cas, plutôt que de parler de deux versions, nous parlons de deux « saveurs » de ConTEXt. Mais qu'on les appelle d'une manière ou d'une autre, le fait est qu'un document préparé pour Mark II peut ne pas être compatible d'une compilation avec Mark IV et vice versa ; et si le document mélange les deux versions, il est fort probable qu'il ne se compilera bien avec aucune d'entre elles ; ce qui signifie que l'auteur du fichier source doit commencer par décider s'il l'écrit pour Mark II ou Mark IV.

Si nous avons à travailler avec différentes versions de ConTEXt, une bonne astuce pour facilement distinguer les versions des fichiers sources consiste à utiliser une extension différente dans le nom des fichiers. Ainsi, par exemple, mes fichiers écrits pour Mark II sont nommés « .mkii » et ceux écrits pour Mark IV sont nommés « .mkiv ». Il est vrai que ConTEXt s'attend à ce que tous les fichiers sources aient l'extension « .tex », mais vous pouvez changer l'extension tant que lorsque vous invoquez un fichier, vous indiquez explicitement son extension, si elle n'est pas celle par défaut.

La distribution de ConTEXt que vous installez à partir de leur wiki, « ConTEXt Standalone », inclut les deux versions, et pour éviter toute confusion —je suppose— propose une commande distincte pour compiler avec chacune d'entre elles. Mark II compile avec la commande « `texexec` » et Mark IV avec la commande « `context` ».

En réalité, aussi bien « `context` » que « `texexec` » sont des *scripts* qui lancent, avec différentes options, « `mtxrun` » qui, à son tour, est un *script* Lua.

A ce jour, Mark II est gelé et Mark IV est toujours en cours de développement, ce qui signifie que les nouvelles versions de Mark II ne sont publiées que lorsque des bogues ou des erreurs sont détectés, tandis que les nouvelles versions de Mark IV sont publiées régulièrement ; parfois même deux ou trois par mois ; bien que dans la plupart des cas, ces « nouvelles versions » n'introduisent pas de changements notables dans le langage, et se limitent à améliorer d'une manière ou d'une autre l'implémentation de bas niveau d'une commande, ou à mettre à jour l'un des nombreux manuels qui sont inclus dans la distribution. Néanmoins, si nous avons installé la version de développement — qui est celle que je recommande et celle qui est installée par défaut avec « ConTeXt Standalone » —, il est approprié de mettre à jour notre installation de temps en temps (voir l'annexe ?? concernant la mise à jour de la version installée de « ConTeXt Standalone »).

LMTX et autres implémentations alternatives de Mark IV

Les développeurs de ConTeXt sont soucieux de la qualité du logiciel et n'ont cessé de faire évoluer Mark IV ; de nouvelles versions sont testées et expérimentées. Celles-ci, en général, ne diffèrent de Mark IV que sur très peu de points, et ne présentent pas d'incompatibilité de compilation comme cela existe entre Mark IV et Mark II, ce qui traduit la maturité du langage du point de vue utilisateur.

Ainsi, quelques variantes de Mark IV ont été développées, appelées respectivement Mark VI, Mark IX et Mark XI. Je n'ai pu trouver qu'une petite référence à Mark VI dans le wiki de ConTeXt où il est indiqué que sa seule différence avec Mark IV est la possibilité de définir des commandes en assignant aux paramètres non pas un nombre, comme c'est traditionnel dans TeX, mais un nom, comme cela se fait habituellement dans presque tous les langages de programmation.

Plus important que ces petites variantes —je pense— est l'apparition dans l'univers de ConTeXt d'une nouvelle version, appelée LMTX, nom qui est un acronyme pour luametaTeX : un nouveau *moteur* de TeX qui est une version simplifiée et optimisée de LuaTeX, développé en vue d'économiser les ressources informatiques et d'offrir une solution TeX aussi minimalistique que possible ; c'est-à-dire que LMTX nécessite moins de place sur disque dur, moins de mémoire et moins de puissance de traitement que ConTeXt Mark IV.

LMTX a été présenté au printemps 2019 et l'on suppose qu'il n'impliquera aucune altération externe du langage Mark IV. Pour l'auteur du document, il n'y aura aucune différence dans la conception ; mais au moment de la compilation, vous pouvez choisir entre compiler avec LuaTeX, ou compiler avec luametaTeX. Une procédure pour attribuer un nom de commande différent à chacune des installations ([section 3](#)) est expliquée dans l'annexe ??, relative à l'installation de ConTeXt.

1.5.2 ConTeXt versus L^AT_EX

Comme le format dérivé de TeX le plus populaire est L^AT_EX la comparaison entre celui-ci et ConTeXt est inévitable.

Il s'agit bien sûr de langages différents mais, d'une certaine manière, liés par leur origine commune TeX ; la parenté est donc similaire à celle qui existe entre, par exemple, l'espagnol et le français : des langues qui partagent une origine commune (le latin) qui utilise des syntaxes *similaires* et de nombreux mots se correspondent assez directement. Mais au-delà de cet air de famille, L^AT_EX et ConTeXt diffèrent dans leur philosophie et leur mise en œuvre, puisque les objectifs initiaux de l'un et de l'autre sont, en quelque sorte, contradictoires.

\LaTeX vise à faciliter l'utilisation de \TeX , en éloignant l'auteur des détails typographiques spécifiques pour l'inciter à se concentrer sur le contenu, et laisser les détails de la composition entre les mains de \LaTeX . En d'autres termes, la simplification de l'utilisation de \TeX est obtenue au prix d'une limitation de son immense flexibilité, par la prédefinition de nombreux formats de base et la réduction du nombre de choix typographiques que l'auteur doit déterminer.

A l'opposé de cette philosophie, ConTeXt est né au sein d'une entreprise dédiée à la composition de documents. Par conséquent, loin d'essayer d'isoler l'auteur des détails de la composition, ce qu'il tente de faire, c'est de lui donner un contrôle absolu et complet sur ceux-ci. Pour ce faire, ConTeXt fournit une interface homogène et cohérente qui reste beaucoup plus proche de l'esprit original \TeX que \LaTeX .

Cette différence de philosophie et d'objectifs fondamentaux se traduit à son tour par une différence de mise en œuvre. Parce que \LaTeX , qui tend à simplifier au maximum, n'a pas besoin d'utiliser toutes les ressources de \TeX . Son cœur est, d'une certaine manière, assez simple. Par conséquent, lorsque vous souhaitez étendre ses possibilités, vous devez construire un *paquet*. Cet ensemble de paquets associé à \LaTeX est à la fois une force et une faiblesse : une force, car l'énorme popularité de \LaTeX , ainsi que la générosité de ses utilisateurs, impliquent que pratiquement tous les besoins qui se présentent ont déjà été soulevés par quelqu'un, et qu'il existe un paquet qui y réponde ; mais aussi une faiblesse, car ces paquets sont souvent incompatibles entre eux, et leur syntaxe n'est pas toujours homogène, ce qui signifie que l'utilisation de \LaTeX exige une plongée continue dans les milliers de paquets existants pour trouver ceux dont nous avons besoin et les faire fonctionner ensemble.

Contrairement à la simplicité du noyau de \LaTeX et son extensibilité par le biais de paquets, ConTeXt est conçu pour intégrer et rendre accessibles toutes — ou presque toutes — les possibilités typographiques de \TeX , de sorte que sa conception est beaucoup plus monolithique, mais, en même temps, il est aussi plus modulaire : le noyau ConTeXt permet de faire presque tout et il est garanti qu'il n'y aura pas d'incompatibilités entre les différentes commandes, il n'y a pas besoin de rechercher les extensions dont vous avez besoin (elles sont déjà présentes), et la syntaxe du langage est homogène entre les différents commandes.

Il est vrai que ConTeXt propose des *modules* d'extension dont on pourrait considérer qu'ils ont une fonction similaire à celle des paquets de \LaTeX , mais la vérité est que la fonction des deux est très différente : les modules de ConTeXt sont conçus exclusivement pour accueillir des fonctionnalités supplémentaires qui, parce qu'ils sont en phase expérimentale, n'ont pas encore été incorporés dans le noyau, ou pour permettre à des développeurs en dehors de l'équipe de développement de ConTeXt de les proposer.

Je ne pense pas que l'une de ces deux *philosophies* puisse être considérée comme préférable à l'autre. La réponse dépend plutôt du profil de l'utilisateur et de ce qu'il souhaite. Si l'utilisateur ne veut pas s'occuper de questions typographiques, mais simplement produire des documents standardisés de très haute qualité, il serait probablement préférable pour lui d'opter pour un système comme \LaTeX ; au contraire, l'utilisateur qui aime expérimenter, ou qui a besoin de contrôler chaque détail de

ses documents, ou qui doit concevoir un design spécial pour un certain document, ferait probablement mieux d'utiliser un système comme ConTeXt, où l'auteur dispose de tous les contrôle ; avec le risque, bien sûr, qu'il ne sache pas correctement l'utiliser.

1.5.3 La logique de travail avec ConTeXt

Lorsque nous travaillons avec ConTeXt, nous commençons toujours par écrire un fichier texte (que nous appellerons *fichier source*), dans lequel nous inclurons, en plus du contenu de notre document final à proprement parler, les instructions (en langage ConTeXt) qui indiquent exactement comment nous voulons que le document soit composé : quel aspect général nous voulons donner à ses pages et paragraphes, quelles marges nous souhaitons appliquer à certains paragraphes spéciaux, quelles types de police doit être utilisé, quels fragments souhaitons nous afficher avec une police différente, etc. Une fois que nous avons écrit le fichier source, depuis un terminal, nous exécuterons le programme « *context* », qui le traitera et, à partir de celui-ci, générera un fichier différent, dans lequel le contenu de notre document aura été formaté selon les instructions qui étaient incluses dans le fichier source. Ce nouveau fichier peut être envoyé à l'imprimante, affiché à l'écran, hébergé sur Internet ou distribué à nos contacts, amis, clients, professeurs, étudiants... bref, à tous ceux pour qui nous avons écrit le document.

C'est-à-dire que lorsqu'il travaille avec ConTeXt, l'auteur agit sur un fichier dont l'apparence n'a rien à voir avec celle du document final : le fichier avec lequel l'auteur travaille directement est un fichier texte qui n'est pas formaté typographiquement. À cet égard, son fonctionnement est très différent de celui des programmes dits de *traitement de texte*, qui affichent l'aspect final du document édité au fur et à mesure de sa saisie. Pour ceux qui sont habitués aux *traitements de texte*, le fonctionnement de l'application peut sembler étrange au début, et il peut même falloir un certain temps pour s'y habituer. Cependant, une fois que vous vous y serez habitué, vous comprendrez que cette autre façon de travailler, faisant la différence entre le fichier de travail et le résultat final, est en fait un avantage pour de nombreuses raisons, parmi lesquelles je soulignerai, sans ordre particulier, les suivantes :

1. car les fichiers texte sont plus « légers » à manipuler que les fichiers binaires des traitements de texte et que leur édition nécessite moins de ressources informatiques ; ils sont moins sujets à la corruption et ne deviennent pas illibres si la version du programme avec lequel ils ont été créés change. Ils sont également compatibles avec n'importe quel système d'exploitation et peuvent être édités avec de nombreux éditeurs de texte, de sorte que pour travailler avec eux, il n'est pas nécessaire de disposer d'un logiciel d'édition particulier : n'importe quel autre programme d'édition de texte fera l'affaire, et chaque système d'exploitation informatique propose un voire des programmes d'édition de texte.
2. car la différenciation entre le document de travail et le document final permet de distinguer ce qui est le contenu réel du document de ce qui sera son apparence, permettant à l'auteur de se concentrer sur le contenu dans la phase de création, et sur l'apparence dans la phase de composition.

3. car il vous permet de modifier très rapidement et très précisément l'apparence du document, puisque celle-ci est déterminée par des commandes facilement identifiables.
4. car cette facilité à changer l'apparence, d'autre part, permet de générer facilement plusieurs versions différentes à partir d'un seul contenu : par exemple une version optimisée pour l'impression sur papier, et une autre pour l'affichage sur écran, ajustée à la taille de celui-ci et, peut-être, incluant des hyperliens qui n'ont pas d'utilité dans un document imprimé sur papier.
5. car il est également facile d'éviter les erreurs typographiques courantes dans les traitements de texte comme, par exemple, l'extension de l'italique au-delà du dernier caractère à utiliser, les erreurs d'application de style..
6. car, puisque le fichier de travail ne sera pas distribué et qu'il est « pour nos yeux seulement », il est possible d'incorporer des annotations et des observations, des commentaires et des avertissements pour nous-mêmes, pour des révisions ou des versions futures, avec la tranquillité d'esprit de savoir qu'ils n'apparaîtront pas dans le fichier formaté qui sera distribué.
7. car la qualité que l'on peut obtenir en traitant simultanément l'ensemble du document est bien supérieure à celle que l'on peut obtenir avec un programme qui doit prendre des décisions typographiques à la volée, au fur et à mesure de la rédaction du document.
8. etcétera.

Tout cela signifie que, d'une part, lorsque l'on travaille avec ConTeXt, une fois que l'on a pris le coup de main, on est plus efficace et productif, et que, d'autre part, la qualité typographique que l'on obtiendra est bien supérieure à celle que l'on obtiendrait avec les *logiciels de traitement de texte*. Et s'il est vrai que, en comparaison, ces derniers sont plus faciles à utiliser, en réalité ils ne le sont *pas beaucoup*. Car s'il est vrai que ConTeXt se compose, comme je l'ai déjà dit, d'environ 3500 instructions, un utilisateur normal n'a pas à toutes les connaître. Pour faire ce que l'on fait habituellement avec les traitements de texte, il suffira de connaître les instructions qui permettent d'indiquer la structure du document, quelques instructions relatives aux ressources typographiques courantes, comme le gras ou l'italique, et, éventuellement, comment générer une liste, ou une note de bas de page. Au total, pas plus de 15 ou 20 instructions nous permettront de faire presque toutes les choses que l'on fait avec un traitement de texte. Le reste des instructions nous permet de faire différentes choses qui, normalement, sont très difficiles voire impossibles à faire avec un logiciel de traitement de texte. Ainsi, si l'apprentissage de ConTeXt est plus difficile que celui d'un logiciel de traitement de texte, c'est parce que l'on peut faire beaucoup plus de choses avec.

1.5.4 Obtenir de l'aide sur ConTeXt

Tant que nous sommes des débutants, le meilleur endroit pour trouver de l'aide sur ConTeXt est sans aucun doute son [wiki](#), qui regorge d'exemples et dispose d'un bon moteur de recherche, même s'il nécessite bien sûr de bien comprendre l'anglais. Nous pouvons aussi chercher de l'aide sur Internet, mais ici le jeu de mots sur lequel repose ConTeXt nous jouera un sale tour car une recherche d'informations sur « contexte » renverra des millions de résultats et la plupart d'entre eux

n'auraient aucun rapport avec ce que nous recherchons. Pour rechercher des informations sur ConTeXt, vous devez ajouter quelque chose au nom « context » ; par exemple, « tex », « luatex », « Mark IV », « Hans Hagen » (un des créateurs de ConTeXt), « Pragma ADE », ou quelque chose de similaire (par exemple une autre commande souvent utilisée dans le cas de figure qui vous préoccupe). Il peut également être utile de rechercher des informations par le nom wiki : « contextgarden ».

Après en avoir appris un peu plus sur ConTeXt, et si l'on maîtrise bien l'anglais, on peut consulter l'un des nombreux documents inclus dans « ConTeXt Standalone » ou demander de l'aide :

- soit sur [TeX – LaTeX Stack Exchange](#) et en particulier [les questions tagguées « ConTeXt »](#)
- soit sur la liste de diffusion propre à ConTeXt [NTG-context](#) et son moteur de recherche.

Cette dernière liste diffusion implique les personnes les plus compétents sur ConTeXt, mais les règles d'une bonne éducation de « cybercitoyen » exigent qu'avant de poser une question, on ait essayé par tous les moyens de trouver la réponse par soi-même dans les documentations déjà existantes.

Chapitre 2

Notre premier fichier source

Table of Contents : [2.1 Préparation de l'expérience outils nécessaires](#) ; [2.2 L'expérience elle-même](#) ; [A Rédaction du fichier source](#) ; [B Encodage du fichier](#) ; [C Regardons le contenu de notre premier fichier source pour ConTeXt](#) ; [D Traitement du document source](#) ; [2.3 La structure de notre fichier d'exemple](#) ; [2.4 Quelques détails supplémentaires sur la façon d'exécuter « context »](#) ; [2.5 Traitement des erreurs](#) ;

Ce chapitre est consacré à la mise en oeuvre de notre première expérience. Il expliquera la structure de base d'un document ConTeXt ainsi que les meilleures stratégies pour faire face aux éventuelles erreurs.

2.1 Préparation de l'expérience outils nécessaires

Pour écrire et compiler un premier fichier source, nous devrons avoir les outils suivants installés sur notre système.

1. **un éditeur de texte** pour écrire notre fichier de test. Il existe de nombreux éditeurs de texte et il est difficilement concevable qu'un système informatique n'en ait pas déjà un d'installé. Nous pouvons utiliser n'importe lequel d'entre eux : il existe des systèmes simples, d'autres complexes, des puissants, d'autres simples, des payants, des gratuits, des spécialisés pour TeX, des généralistes, etc. Si nous avons l'habitude d'utiliser un éditeur spécifique, il est préférable de poursuivre avec lui ; si nous n'avons pas l'habitude de travailler avec des éditeurs de texte, mon conseil est, dans un premier temps, de choisir un éditeur simple, afin de ne pas ajouter à la difficulté de l'apprentissage de ConTeXt la difficulté d'apprendre à utiliser l'éditeur. Bien qu'il soit également vrai que, souvent, les programmes les plus difficiles à maîtriser sont aussi les plus puissants.

J'ai écrit ce texte avec GNU Emacs, qui est l'un des éditeurs généralistes les plus puissants et les plus polyvalents qui existent ; il est vrai qu'il a ses particularités et aussi ses détracteurs, mais en général il y a plus de « *Emacs Lovers* » que de « *Emacs Haters* ». Pour travailler avec des fichiers TeX ou l'un de ses dérivés, il

existe une extension pour GNU Emacs, appelée AucTeX, qui fournit à l'éditeur quelques fonctionnalités supplémentaires très intéressantes, même si AucTeX est plus développé pour L^AT_EX que pour ConTeXt. GNU Emacs en combinaison avec AucTeX peut être une bonne option si l'on ne sait pas quel éditeur choisir ; tous deux sont des programmes à code source ouvert, et ils sont disponibles pour tous les systèmes d'exploitation. En fait, dire que GNU Emacs est un *logiciel libre* est un euphémisme, car ce programme incarne mieux que tout autre l'esprit de ce qu'est et signifie le *logiciel libre*. Après tout, son principal développeur était RICHARD STALLMAN, fondateur et idéologue du projet GNU et de la *Free Software Foundation*.

En plus de GNU Emacs + AucTeX, *Scite* et *TexWorks* sont d'autres bonnes options si vous ne savez pas quel éditeur choisir. Le premier, bien qu'il s'agisse d'un éditeur à usage généraliste, non conçu spécifiquement pour travailler avec des fichiers ConTeXt, est facilement personnalisable et, comme c'est l'éditeur généralement utilisé par les développeurs de ConTeXt « ConTeXt Standalone » contient les fichiers de configuration de cet éditeur conçus et utilisés par HANS HAGEN lui-même. *TexWorks* est, quant à lui, un éditeur de texte rapide, spécialisé dans le traitement des fichiers T_EX et de ses langages dérivés. Il est assez facile à configurer pour fonctionner avec ConTeXt et « ConTeXt Standalone » prévoit également de fournir des fichiers configurations.

Qu'il s'agisse d'un éditeur ou d'un autre, ce qu'il ne faut pas faire, c'est utiliser, comme éditeur de texte, un *logiciel de traitement de texte* tel que, par exemple, OpenOffice Writer ou Microsoft Word. Ces programmes, qui sont à mon avis trop lents et trop lourds, peuvent certes enregistrer un fichier en « texte pur », mais ils ne sont pas conçus pour cela et nous finirions probablement par enregistrer notre fichier dans un format binaire incompatible avec ConTeXt.

2. **Une distribution ConTeXt** pour traiter notre fichier de test. S'il existe déjà une installation T_EX (ou L^AT_EX) sur votre système, il est possible qu'une version de ConTeXt soit déjà installée. Pour le vérifier, il suffit d'ouvrir un terminal et de taper dans celui-ci

```
$ context --version
```

NOTA ceux pour qui l'utilisation du terminal est nouvelle, les deux premiers caractères que j'ai indiqué (« \$ ») n'ont pas à être tappé dans le terminal par l'utilisateur. Je les utilise pour représenter ce qu'on appelle l'*invite* du terminal (le prompt en anglais), qui indique que le terminal attend nos instructions.

Si une version de ConTeXt est déjà installée, vous devriez obtenir un résultat similaire au suivant :

```
$ context --version
mtx-context      | ConTeXt Process Management 1.03
mtx-context      |
mtx-context      | main context file: /usr/share/texmf/tex/context/base/mkiv/context
mtx-context      | current version: 2020.03.10 14:44
mtx-context      | main context file: /usr/share/texmf/tex/context/base/mkiv/context
mtx-context      | current version: 2020.03.10 14:44
```

Dans la dernière ligne, nous sommes informés de la date à laquelle la version installée a été publiée. Si elle est très ancienne, nous devons le mettre à jour ou installer une nouvelle version. Je recommande d'installer la distribution appelée « ConTeXt Standalone » dont les instructions d'installation se trouvent sur le [wiki de ConTeXt](#). Les indications sont également incluses dans l'annexe ??.

3. **Un programme de visualisation de fichier PDF** afin de visualiser le résultat de notre expérience à l'écran. Sur les systèmes Windows et Mac OS, la visionneuse omniprésente est Adobe Acrobat Reader. Il n'est pas installé par défaut (ou ne l'était pas lorsque j'ai cessé d'utiliser Microsoft Windows, il y a plus de 15 ans). L'installation se fait la première fois que vous essayez d'ouvrir un fichier PDF, il est donc généralement déjà installé. Sur les systèmes Linux/Unix, il n'y a pas de version mise à jour d'Acrobat Reader, mais il n'est pas nécessaire non plus, car il existe littéralement des dizaines de très bons visualiseurs de PDF gratuits. De plus, dans ces systèmes, il y en a presque toujours un installé par défaut. Mon préféré, pour sa vitesse et sa facilité d'utilisation, est MuPDF ; bien qu'il ait quelques inconvénients comme, par exemple, de ne pas montrer l'index des signets, de ne pas permettre les recherches de texte qui incluent des caractères inexistant dans l'alphabet anglais (comme les voyelles accentuées ou les eñes) ou de ne pas permettre de sélectionner le texte, d'envoyer le document à l'imprimante ; c'est juste un visualiseur, mais très rapide et très confortable. Lorsque j'ai besoin de certains de ces fonctionnalités absentes de MuPDF, j'utilise généralement Okular ou qPdfView. Mais, encore une fois, la question est une affaire de goût : chacun peut choisir celui qu'il préfère.

Nous pouvons choisir l'éditeur, nous pouvons choisir le visualisateur de PDF, nous pouvons choisir la distribution ConTeXt... Bienvenue dans le monde du *logiciel libre* !

2.2 L'expérience elle-même

A. Rédaction du fichier source

Si nous disposons déjà des outils mentionnés dans la section précédente, nous devons ouvrir notre éditeur de texte et créer un fichier appelé « la-maison-sur-le-port.tex » pour notre exemple. Comme contenu du fichier, nous allons écrire ce qui suit :

```
% Première ligne du document

\mainlanguage[fr]      % Langue français

\setuppapersize[S5]    % Format du papier

\setupbodyfont        % Police = Latin Modern, 12 points
    [modern,18pt]

\setuphead            % Format des titres de chapitre
    [chapter]
    [style=\bfc]

\starttext             % Début du contenu du document

\startchapter           [title=La maison sur le port]

Il y avait des      chansons
Les hommes          venaient y boire et rêver
Dans la maison      sur le port
Où les filles        riaient fort
Où le vin faisait   chanter chanter chanter

Les pêcheurs         vous le diront
Ils y venaient       sans façon
Avant de partir     retirer leurs filets
Ils venaient         se réchauffer près de nous
Dans la maison       sur le port

\stopchapter

\stoptext             % Fin du document
```



Durant l'écriture, certains aspects n'ont aucune importance, notamment si vous ajoutez ou supprimez des espaces blancs ou des sauts de ligne. Ce qui est important, c'est que chaque mot suivant le caractère « \ » soit écrit très exactement de la même façon qu'il l'est dans l'exemple, ainsi que le contenu des crochets. Il peut y avoir des variations dans le reste.

B. Encodage du fichier

Une fois le texte précédent écrit, nous enregistrons le fichier sur le disque. Cela n'est dorénavant qu'une vérification à faire, mais il faut nous assurer que l'encodage du fichier est bien UTF-8. Cet encodage est aujourd'hui la norme et constitue l'encodage par défaut sur la plupart des systèmes Linux/Unix. Néanmoins, je ne sais pas si c'est la même chose sous Mac OS ou Windows et il est encore tout à fait possible que l'encodage ANSI soit utilisé. En tout cas, si nous ne sommes pas sûrs, depuis l'éditeur de texte lui-même, nous pouvons voir avec quel encodage le fichier sera enregistré et, si nécessaire, le modifier. La manière de procéder dépend, bien entendu, de l'éditeur avec lequel nous travaillons. Dans GNU Emacs, par exemple, en appuyant simultanément sur les touches CTRL-X puis Return suivi de « f », dans la dernière ligne de la fenêtre (que GNU Emacs appelle mini-buffer) un message apparaîtra nous demandant un nouvel encodage et nous informant de l'encodage actuel. Dans les autres éditeurs, nous pouvons généralement accéder à l'encodage dans le menu « Enregistrer sous ».

Après avoir vérifié que l'encodage est correct et enregistré le fichier sur le disque, nous fermerons l'éditeur pour nous concentrer sur l'analyse de ce que nous avons écrit.

C. Regardons le contenu de notre premier fichier source pour ConTeXt

La première ligne commence par le caractère « % ». C'est un caractère réservé qui indique à ConTeXt de ne pas traiter le texte qui le suit et ce jusqu'à la fin de la ligne sur laquelle il se trouve. Cette fonctionnalité est utilisée pour écrire des commentaires dans le fichier source que seul l'auteur pourra lire, car ils ne seront pas incorporés au document final. Dans cet exemple, je l'ai par exemple utilisé pour attirer l'attention sur certaines lignes, en expliquant ce qu'elles font.

Les lignes suivantes commencent par le caractère « \ » qui est un autre des caractères réservés de ConTeXt et indique que ce qui suit est le nom d'une commande. L'exemple comprend plusieurs commandes couramment utilisées dans un fichier source ConTeXt : la langue dans laquelle le document est écrit, le format du papier, la police à utiliser dans le document et la mise en forme appliquée aux titres de chapitres. Plus tard, dans d'autres chapitres, nous détailleront ces commandes, pour le moment je veux juste que le lecteur voit à quoi elles ressemblent : elles commencent toujours par le caractère « \ », suivi du nom de la commande, et ensuite, entre parenthèses ou accolades, selon le cas, les données dont la commande a besoin pour produire ses effets. Entre le nom de la commande et les crochets ou accolades qui l'accompagnent, il peut y avoir des espaces vides ou des sauts de ligne. C'est à l'auteur de choisir la façon dont le code source est le plus clair et lisible.

Sur la 9^{ème} ligne de notre exemple (je ne compte que les lignes qui ont du texte) se trouve la commande importante `\starttext` : elle indique à ConTeXt que le contenu du document commence à partir de cet endroit ; et à la dernière ligne de notre exemple, nous voyons la commande `\stoptext` qui indique la fin du contenu. Tout ce qui suit cette dernière commande ne sera pas traité. Ce sont deux commandes

très importantes sur lesquelles je reviendrai très bientôt. Entre les deux se trouve donc le contenu à proprement parler de notre document qui, dans notre exemple, consiste en la première strophe de la chanson "« La maison sur le port », dont les paroles sont de AMALIA RODRIGUES, et qui a été reprise notamment par SANSEVERINO. Je l'ai écrit en prose afin de mieux observer le formatage des paragraphes effectué par ConTeXt.

D. Traitement du document source

Pour l'étape suivante, après s'être assuré que ConTeXt a été correctement installé dans notre système, nous devons ouvrir un terminal dans le répertoire où se trouve notre fichier « la-maison-sur-le-port.tex ».

De nombreux éditeurs de texte vous permettent de compiler le document sur lequel vous travaillez sans ouvrir un terminal. Cependant, la procédure *canonique* pour traiter un document avec ConTeXt implique de le faire à partir d'un terminal, en exécutant directement le programme. Je vais procéder de cette manière (ou supposer qu'il en est ainsi) tout au long de ce document pour plusieurs raisons ; la première est que je n'ai aucun moyen de savoir avec quel éditeur chaque lecteur travaille. Mais le plus important est que, depuis le terminal, nous aurons accès à la *sortie* de « context », c'est à dire les messages émis par le programme.

Si la distribution ConTeXt que nous avons installée est « ConTeXt Standalone », nous devons tout d'abord exécuter le *script* qui indique au terminal les chemins et l'emplacement des fichiers dont ConTeXt a besoin pour fonctionner. Sur les systèmes Linux/Unix, cela se fait en tapant la commande suivante :

```
$ source ~/context/tex/setuptex
```

en supposant que nous avons installé ConTeXt dans un répertoire appelé « context ».

En ce qui concerne l'exécution du *script* qui vient d'être mentionné, voir ce qui est dit dans l'annexe ?? concernant l'installation de « ConTeXt Standalone ».

Une fois que les variables nécessaires à l'exécution de « context » ont été chargées en mémoire, nous pouvons l'exécuter. Cela se fait en tapant dans le terminal :

```
$ context la-maison-sur-le-port
```

Notez que bien que le fichier source s'appelle « la-maison-sur-le-port.tex » dans l'appel à « context » nous avons omis l'extension du fichier. Si nous avions appelé au fichier source, par exemple, « la-maison-sur-le-port.mkiv » (ce que je fais habituellement pour savoir que ce fichier est écrit pour Mark IV), il aurait fallu indiquer expressément l'extension du fichier à compiler en tapant « context la-maison-sur-le-port.mkiv ».

Après avoir exécuté « context » dans le terminal, plusieurs dizaines de lignes s'affichent à l'écran, informant de ce que fait ConTeXt. Les informations s'affichent à une vitesse impossible à suivre par un être humain, mais ne vous inquiétez pas, car en plus de l'écran, ces informations sont également stockées dans un fichier auxiliaire, avec l'extension « .log » qui est généré avec la compilation et que nous pourrons consulter tranquillement plus tard si nécessaire.

Après quelques secondes, si nous avons bien écrit le texte de notre fichier source, sans faire d'erreur grave, l'émission de messages vers le terminal se terminera. Le dernier des messages nous informera du temps nécessaire à la compilation. La première fois qu'un document est compilé, cela prend toujours un peu plus de temps, car ConTEXt doit construire à partir de zéro certains fichiers contenant les informations de notre document. Par la suite ils seront juste réutilisés et complétés pour les compilations suivantes qui iront plus vite. Le message du temps passé indique que la compilation est terminée. Si tout s'est bien passé, trois fichiers supplémentaires apparaîtront dans le répertoire où nous avons exécuté « `context` » :

- `la-maison-sur-le-port.pdf`
- `la-maison-sur-le-port.log`
- `la-maison-sur-le-port.tuc`

Le premier est le résultat de notre traitement, c'est-à-dire : le fichier PDF déjà formaté. Le deuxième est le fichier dans lequel sont stockées toutes les informations qui ont été affichées à l'écran pendant la compilation ; le troisième est un fichier auxiliaire que ConTEXt génère pendant la compilation et qui est utilisé pour construire les index et les références croisées. Pour le moment, si tout a fonctionné comme prévu, nous pouvons supprimer les deux fichiers (« `la-maison-sur-le-port.log` » et « `la-maison-sur-le-port.tuc` »). S'il y a eu un problème, les informations contenues dans ces fichiers peuvent nous aider à localiser la source du problème et à déterminer comment le résoudre.

Si nous n'avons pas obtenu ces résultats, c'est probablement dû à un ou plusieurs de points qui suivent :

- soit nous n'avons pas installé correctement notre distribution ConTEXt, auquel cas en tapant la commande « `context` » dans le terminal, un message « commande inconnue » sera apparu.
- soit notre fichier n'a pas été encodé en UTF-8 et cela a généré une erreur de compilation.
- soit peut-être que la version de ConTEXt installée sur notre système est Mark II. Dans cette version, vous ne pouvez pas utiliser l'encodage UTF-8 sans l'indiquer explicitement dans le fichier source lui-même. Nous pourrions corriger le fichier source pour qu'il compile bien, mais, puisque cette introduction se réfère à Mark IV, cela n'a pas de sens de continuer à travailler avec Mark II : la meilleure chose à faire est d'installer « ConTEXt Standalone ».
- Soit nous avons fait une erreur en écrivant dans le fichier source le nom de certaines commandes ou leurs données associées.

Si après l'exécution de « `context` », le terminal a commencé à émettre des messages, mais s'est ensuite arrêté sans que le *prompt* ne réapparaisse, avant de continuer, il faut appuyer sur CTRL-X pour interrompre l'exécution de ConTEXt qui a été interrompue par l'erreur.

En cas de problème, nous devrons donc vérifier chacune de ces possibilités, et les corriger, jusqu'à ce que la compilation se déroule correctement.

1 La maison sur le port

Il y avait des chansons Les hommes venaient
y boire et rêver Dans la maison sur le port Où
les filles riaient fort Où le vin faisait chanter
chanter chanter

Les pêcheurs vous le diront Ils y venaient sans
façon Avant de partir retirer leurs filets Ils ve-
naient se réchauffer près de nous Dans la mai-
son sur le port

Figure B.1 La maison sur le port

La [figure B.1](#) montre le contenu de « la-maison-sur-le-port.pdf ». Nous pouvons voir que ConTeXt a numéroté la page, numéroté le chapitre et écrit le texte dans la police indiquée. Il a également réparti le mot « venaient » entre la sixième et la septième ligne, ainsi que le mot « maison » la septième et la guitième ligne. ConTeXt, par défaut, active la césure (division syllabique) des mots afin de répartir les blancs (les espaces vides entre les mots) de façon la plus homogène possible. C'est pourquoi il est si important d'informer ConTeXt de la langue du document, car les modèles de césure varient selon la langue. Dans notre exemple, c'est l'objectif de la première commande du fichier source (`\mainlanguage[fr]`).

En bref : ConTeXt a transformé le fichier source et a généré un fichier dans lequel nous avons un document formaté selon les instructions qui étaient incluses dans le fichier source. Les commentaires en ont disparu et, en ce qui concerne les commandes, ce que nous avons maintenant n'est pas leur nom, mais le résultat de leur application par ConTeXt.

2.3 La structure de notre fichier d'exemple

Dans un projet aussi simple que notre exemple, développé dans un seul fichier source, la structure de celui-ci est très simple et est marquée par les commandes `\starttext ... \stoptext`. Tout ce qui se trouve entre la première ligne du fichier et la commande `\starttext` constitue le *préambule*. Le contenu du document lui-même est inséré entre les commandes `\starttext` et `\stoptext`. Dans notre exemple, le préambule comprend quatre commandes de configuration globale : une pour indiquer la langue de notre document (`\mainlanguage`), une autre pour indiquer la taille des pages (`\setuppapersize`) qui dans notre cas est « S5 », représentant les proportions d'un écran d'ordinateur, une troisième commande (`\setupbodyfont`) qui nous permet d'indiquer la police de caractère et sa taille, et une quatrième (`\setuphead`) qui nous permet de configurer l'apparence des titres des chapitres.

Le corps du document est encadré par les commandes `\starttext` et `\stoptext`. Ces commandes indiquent, respectivement, le point de départ et le point final du texte à traiter : entre elles, nous devons inclure tout le texte que nous voulons que ConTEXt traite, ainsi que les commandes qui ne doivent pas affecter le document entier mais seulement des fragments de celui-ci. Pour le moment, nous devons supposer que les commandes `\starttext` et `\stoptext` sont obligatoires dans tout document ConTEXt, bien que plus tard, en parlant des projets multifichiers (section ??) nous verrons qu'il y a quelques exceptions.

2.4 Quelques détails supplémentaires sur la façon d'exécuter « context »

La commande « context » avec laquelle nous avons procédé au traitement de notre premier fichier source est, en fait, un *script LUA*, c'est-à-dire : un petit programme LUA qui, après avoir effectué quelques vérifications et opérations, appelle *LuaTeX* pour traiter le fichier source.

Nous pouvons appeler « context » avec plusieurs options. Les options sont saisies immédiatement après le nom de la commande, précédées de deux traits d'union. Si nous voulons saisir plus d'une option, nous les séparons par un espace blanc. L'option « `help` » nous donne une liste de toutes les options, avec une brève explication de chacune d'elles :

```
$ context --help
```

Parmi les options les plus intéressantes, citons les suivantes :

interface Comme je l'ai dit dans le chapitre d'introduction, l'interface de ConTeXt est traduite en plusieurs langues. Par défaut, c'est l'interface anglaise qui est utilisée, mais cette option nous permet de lui demander d'utiliser la version néerlandaise (nl), française (fr), italienne (it), allemande (de) ou roumaine (ro).

purge, purgeall Supprime les fichiers auxiliaires générés pendant le traitement.

result=Name indique le nom que doit porter le fichier PDF résultant. Par défaut, ce sera le même que le fichier source à traiter, avec l'extension « `.pdf` ».

usemodule=list Charge les modules qui sont indiqués avant d'exécuter ConTeXt (un module est une extension de ConTeXt, qui ne fait pas partie de son noyau, et qui lui fournit une utilité supplémentaire).

useenvironment=list Charge les fichiers d'environnement qui sont spécifiés avant de lancer ConTeXt (un fichier d'environnement est un fichier contenant des instructions de configuration).

version indique la version de ConTeXt.

help affiche des informations d'aide sur les options du programme.

noconsole Supprime l'envoi de messages à l'écran pendant la compilation. Toutefois, ces messages seront toujours enregistrés dans le fichier « `.log` ».

nonstopmode Exécute la compilation sans s'arrêter sur les erreurs. Cela ne signifie pas que l'erreur ne se produira pas, mais que lorsque ConTeXt rencontre une erreur, même si elle est récupérable, il continuera la compilation jusqu'à ce qu'elle se termine ou jusqu'à ce qu'il rencontre une erreur irrécupérable.

batchmode Il s'agit d'une combinaison des deux options précédentes. Il fonctionne sans interruption et omet les messages à l'écran.

Pour les premières utilisation et pour l'apprentissage de ConTeXt, je ne pense pas que ce soit une bonne idée d'utiliser les trois dernières options, car lorsqu'une erreur se produit, nous n'aurons aucune idée de l'endroit où elle se trouve ou de ce qui l'a produite. Et, croyez-moi chers lecteurs, tôt ou tard, une erreur de compilation se produira.

2.5 Traitement des erreurs

En travaillant avec ConTEXt, il est inévitable que, tôt ou tard, des erreurs se produisent dans la compilation. En gros, nous pouvons regrouper les erreurs dans l'une des quatre catégories suivantes :

1. **Erreurs de frappe.** Elles se produisent lorsque nous orthographions mal le nom d'une commande. Dans ce cas, nous envoyons au compilateur une commande qu'il ne comprend pas. Par exemple, lorsque, au lieu d'écrire la commande `\TeX`, nous écrivons `\Tex` avec le « X » final en minuscule, puisque ConTEXt fait la différence entre les majuscules et les minuscules et considère donc que « TeX » et « Tex » sont des mots différents ; ou si les options utilisée pour une commande, au lieu de les mettre entre crochets, sont mises entre accolades, ou si nous essayons d'utiliser un des caractères réservés comme s'il s'agissait d'un caractère normal, etc.
2. **Erreurs par omission.** Dans ConTEXt il y a des instructions qui démarrent une tâche, dont il faut indiquer explicitement quand la fermer ; comme le caractère réservé « \$ » qui active le mode mathématique, qui est maintenu jusqu'à ce qu'on le désactive, et si on oublie de le désactiver, une erreur sera générée dès qu'on trouvera un texte ou une instruction qui n'a pas de sens dans le mode mathématique. Il en va de même si nous commençons un bloc de texte au moyen du caractère réservé « { » ou d'une commande `\startUnTruc` et que, par la suite, la fermeture explicite n'est pas trouvée (« } » ou `\stopUnTruc`).
3. **Erreurs de conception.** J'appelle ainsi les erreurs qui se produisent lorsque vous appelez une commande qui nécessite certains arguments, sans les fournir, ou lorsque la syntaxe d'appel de la commande n'est pas correcte.
4. **Erreurs situationnelles.** Certaines commandes sont destinées à ne fonctionner que dans certains contextes ou environnements, et sont donc inconnues en dehors de ceux-ci. Cela se produit, en particulier, avec le mode mathématique : certaines commandes ConTEXt ne fonctionnent que lors de l'écriture de formules mathématiques et si elles sont appelées dans d'autres contextes, elles génèrent une erreur.

Que faire lorsque « `context` » nous avertit, pendant la compilation, qu'une erreur s'est produite ? La première chose est, évidemment, d'identifier quelle est l'erreur. Pour ce faire, nous devrons parfois analyser le fichier « `.log` » généré pendant la compilation ; mais encore plus souvent il suffira de remonter dans les messages produits par « `context` » dans le terminal où il est exécuté.

```

tex error      > tex error on line 14 in file la-maison-sur-le-port_bug.tex:
! Undefined control sequence

1.14 \starttext
                  % Début du contenu du document

4
5   \setuppapersize[S5]  % Format du papier
6
7   \setupbodyfont      % Police = Latin Modern, 12 points
8     [modern,18pt]
9
10  \setuphead         % Format des titres de chapitre
11    [chapter]
12    [style=\bfc]
13
14 >> \starttext        % Début du contenu du document
15
16  \startchapter
17    [title=La maison sur le port]
18
19  Il y avait des    chansons
20  Les hommes        venaient y boire et rêver
21  Dans la maison    sur le port
22  Où les filles      riaient fort
23  Où le vin faisait chanter chanter chanter
24

mtx-context      | fatal error: return code: 256

```

Par exemple, si dans notre fichier de test, « `la-maison-sur-le-port.tex` », par erreur, au lieu de `\starttext` nous avions écrit `\starttext` (avec un seul « `t` »), ce qui, par ailleurs, est une erreur très courante, lors de l'exécution de « `context la-maison-sur-le-port` », lorsque la compilation était arrêtée, dans l'écran du terminal nous pouvions voir l'information montrée dans ci-dessus.

Nous pouvons y voir les lignes de notre fichier source numérotées, et à l'une d'entre elles, dans notre cas la ligne 14, entre le numéro et le texte de la ligne le compilateur a ajouté « `>>` » pour indiquer que c'est dans cette ligne qu'il a trouvé l'erreur. Le numéro de la ligne est également indiqué plus haut, avant l'affichage des lignes, dans une ligne commençant par « `tex error` ». Le fichier « `la-maison-sur-le-port.log` » nous donnera plus d'indices. Dans notre exemple, il ne s'agit pas d'un très gros fichier, car la source que nous compilions est très petite ; dans d'autres cas, il peut contenir une quantité écrasante d'informations. Mais nous devons nous y plonger. Si nous ouvrons « `la-maison-sur-le-port.log` » avec un éditeur de texte, nous verrons que ce fichier enregistre tout ce que fait ConTeXt. Nous devons y chercher une ligne qui commence par un avertissement d'erreur « `tex error` », pour cela nous pouvons utiliser la fonction de recherche de texte de l'éditeur. Nous trouverons les lignes d'erreur suivantes :

```
tex error      > tex error on line 14 in file la-maison-sur-le-port_bug.tex:  
! Undefined control sequence  
  
1.14 \starttext  
          % Début du contenu du document
```

Note : La première ligne informant de l'erreur, dans le fichier « la-maison-sur-le-port.log » est très longue. Pour que cela soit présentable ici, en tenant compte de la largeur de la page, j'ai supprimé une partie du chemin indiquant l'emplacement du fichier.

Si nous prêtons attention aux trois lignes du message d'erreur, nous voyons que la première nous indique à quel numéro de ligne l'erreur s'est produite (ligne 14) et de quel type d'erreur il s'agit : « Undefined control sequence », ou, ce qui revient au même : « Unknown control sequence », c'est-à-dire une commande inconnue. Les deux lignes suivantes du fichier journal nous montrent la ligne 14, qui commence à l'endroit où l'erreur s'est produite. Donc il n'y a pas de doute, l'erreur est dans `\starttext`. Nous le lirons attentivement et, avec de l'attention et de l'expérience, nous nous rendrons compte que nous avons écrit « starttext » et non « starttext » (avec un double « t »).

Pensez que les ordinateurs sont très bons et très rapides pour exécuter des instructions, mais très maladroits pour lire nos pensées, et que le mot « starttext » n'est pas le même que « startttext ». Dans le second cas, le programme sait comment l'exécuter ; dans le premier cas, il ne sait pas quoi faire.

D'autres fois, la localisation de l'erreur ne sera pas aussi facile. En particulier lorsque l'erreur est qu'une tâche a été lancée et que sa fin n'a pas été expressément spécifiée. Parfois, au lieu de chercher l'expression « tex error » dans le fichier « .log », vous devez chercher un astérisque. Ce caractère au début d'une ligne du fichier journal représente, non pas une erreur fatale, mais un avertissement. Et les avertissements peuvent être utiles pour localiser l'erreur.

Et si les informations du fichier « .log » ne sont pas suffisantes, il faudra aller, petit à petit, localiser l'endroit de l'erreur. Une bonne stratégie pour cela consiste à changer l'emplacement de la commande `\stoptext` dans le fichier source. Rappelez-vous que ConTeXt arrête de traiter le texte dès qu'il trouve cette commande. Par conséquent, si, dans mon fichier source, j'écris, plus ou moins à la hauteur du milieu, un `\stoptext` et que je compile, seule la première moitié sera traitée ; si l'erreur se répète, je saurai qu'elle se trouve dans la première moitié du fichier source, si elle ne se répète pas, cela signifie que l'erreur se trouve dans la deuxième moitié... et ainsi, petit à petit, en changeant l'emplacement de la commande `\stoptext`, nous pouvons localiser l'emplacement de l'erreur.

Une autre astuce consiste à mettre en commentaires le paquets de lignes douteuses avec le caractère « % » (certain éditeur de texte propose une fonction pour commenter et décommenter tout un paquet de ligne automatiquement).

Une fois que nous l'avons localisée, nous pouvons essayer de la comprendre et de la corriger ou, si nous ne pouvons pas comprendre pourquoi l'erreur se produit, au moins, ayant localisé le point où elle se trouve, nous pouvons essayer d'écrire les choses d'une manière différente pour éviter que l'erreur se reproduise.i

Ce dernier point, bien sûr, uniquement si nous sommes les auteurs ; si nous nous limitons à composer le texte de quelqu'un d'autre, nous ne pourrons pas le modifier et nous devrons continuer à enquêter jusqu'à ce que nous découvrions les raisons de l'erreur et sa possible solution.

Dans la pratique, lorsqu'on crée un document relativement volumineux avec ConTeXt, ce que l'on fait habituellement, c'est de le compiler de temps en temps, au fur et à mesure de la rédaction du document, de sorte que si une erreur se produit, nous savons plus ou moins clairement quelle est la partie du document qui vient d'être introduite depuis la précédente compilation et qui engendre la nouvelle erreur.

Chapitre 3

Les commandes et autres concepts fondamentaux de ConTeXt

Table of Contents : [3.1 Les caractères réservés de ConTeXt](#); [3.2 Les commandes à proprement parler](#); [3.3 Périmètre des commandes](#); [3.3.1 Les commandes qui nécessite ou pas une périmètre d'application](#); [3.3.2 Commandes nécessitant d'indiquer leur début et fin d'application \(environnements\)](#); [3.4 Options de fonctionnement des commandes](#); [3.4.1 Commandes qui peuvent fonctionner de différentes façon distinctes](#); [3.4.2 Les commandes qui configurent comment d'autres commandes fonctionnent \(`\setupQuelqueChose`\)](#); [3.4.3 Définir des versions personnalisée de commande configurables \(`\defineQuelqueChose`\)](#); [3.5 Résumé sur la syntaxe des commandes et des options, et sur l'utilisation des crochets et des accolades lors de leur appel.](#); [3.6 La liste officielle des commandes ConTeXt](#); [3.7 Définir de nouvelles commandes](#); [3.7.1 Mécanisme général pour définir de nouvelles commandes](#); [3.7.2 Création de nouveaux environnements](#); [3.8 Autres concepts fondamentaux](#); [3.8.1 Groupes](#); [3.8.2 Dimensions](#); [3.9 Méthode d'auto apprentissage pour ConTeXt](#);

Nous avons déjà vu que dans le fichier source, avec le contenu réel de notre futur document formaté, nous insérons les instructions nécessaires pour expliquer à ConTeXt comment nous voulons que notre contenu soit mis en forme. Nous pouvons appeler ces instructions « commandes », « macros » ou « séquences de contrôle ».

Du point de vue du fonctionnement interne de ConTeXt (en fait, du fonctionnement de TeX), il y a une différence entre les *primitives* et les *macros*. Une primitive est une instruction simple qui ne peut pas être décomposée en d'autres instructions plus simples. Une macro est une instruction qui peut être décomposée en d'autres instructions plus simples qui, à leur tour, peuvent peut-être aussi être décomposées en d'autres encore, et ainsi desuite. La plupart des instructions de ConTeXt sont, en fait, des macros. Du point de vue du programmeur, la différence entre les macros et les primitives est importante. Mais du point de vue de l'utilisateur, la question n'est pas si importante : dans les deux cas, nous avons des instructions qui sont exécutées sans que nous ayons besoin de nous préoccuper de leur fonctionnement à un niveau inférieur. Par conséquent, la documentation ConTeXt parle généralement d'une *commande* lorsqu'elle adopte le point de vue de l'utilisateur, et d'un *macro* lorsqu'elle adopte le point de vue du programmeur. Puisque nous ne prenons que la perspective de l'utilisateur dans cette introduction, j'utiliserai l'un ou l'autre terme, les considérant comme synonymes.

⁸ Dans la terminologie informatique, la touche qui affecte l'interprétation du caractère suivant est appelée le « caractère d'échappement ». En revanche, la touche *escape key* des claviers est appelée ainsi car elle génère le caractère 27 en code ASCII, qui est utilisé comme caractère d'échappement dans cet encodage. Aujourd'hui, l'utilisation de la touche Echap est davantage associée à l'idée d'annuler une action en cours.

Les *commandes* sont des ordres donnés au programme ConTeXt pour qu'il fasse quelque chose. Nous contrôlons les performances du programme par leur intermédiaire. Ainsi KNUTH, le père de TeX, utilise le terme de *séquences de contrôle* pour se référer à la fois aux primitives et aux macros, et je pense que c'est le terme le plus précis de tous. Je l'utiliserais lorsque je penserai qu'il est important de distinguer entre *symboles de contrôle* et *mots de contrôle*.

Les instructions de ConTeXt sont essentiellement de deux sortes : les caractères réservés, et les commandes proprement dites.

3.1 Les caractères réservés de ConTeXt

Lorsque ConTeXt lit le fichier source composé uniquement de caractères de texte, puisqu'il s'agit d'un fichier texte, il doit d'une manière ou d'une autre distinguer ce qui est le contenu textuel à mettre en forme, et les instructions qu'il doit exécuter. Les caractères réservés de ConTeXt sont ce qui lui permet de faire cette distinction. En principe, ConTeXt suppose que chaque caractère du fichier source est un texte à traiter, sauf s'il s'agit de l'un des 11 caractères réservés qui doivent être traités comme une *instruction*.

Seulement 11 instructions ? Non. Il n'y a que 11 caractères réservés, mais l'un d'entre eux, le caractère de « \ », a pour fonction de convertir le ou les caractères qui le suivent immédiatement en instruction, rendant ainsi le nombre potentiel de commandes illimité. ConTeXt a environ 3000 commandes (en additionnant les commandes exclusives à Mark II, Mark IV et celles communes aux deux versions).

Les caractères réservés sont les suivants :

\ % { } # ~ | \$ _ ^ &

ConTeXt les interprète de la façon suivante :

\ Ce caractère est le plus important pour nous : il indique que ce qui vient immédiatement après ne doit pas être interprété comme du texte mais comme une instruction. Il est appelé « Caractère d'échappement » ou « Séquence d'échappement » (même s'il n'a rien à voir avec la touche « Esc » que l'on trouve sur la plupart des claviers).⁸

% Indique à ConTeXt que ce qui suit jusqu'à la fin de la ligne est un commentaire qui ne doit pas être traité ou inclus dans le fichier formaté final. L'introduction de commentaires dans le fichier source est extrêmement utile. Cela permet par exemple d'expliquer pourquoi quelque chose a été fait d'une certaine manière, comment tel ou tel effet graphique a été obtenu, garder un rappel d'une idée à compléter ou à réviser, d'une illustration à construire.

Il peut également être utilisé pour aider à localiser une erreur dans le fichier source, puisqu'en commentant une ligne, nous l'excluons de la compilation, et pouvons voir si elle est à l'origine de l'erreur de compilation. Le commentaire peut aussi être utilisé pour stocker deux versions différentes

d'une même macro, et ainsi obtenir des résultats différents après la compilation ; ou pour empêcher la compilation d'un extrait dont nous ne sommes pas sûrs, mais sans le supprimer du fichier source au cas où nous voudrions y revenir plus tard ; ou pour partager des commentaires lors de l'édition en mode collaboratif d'un document... etc.

Avec la possibilité que notre fichier source contienne du texte que personne d'autre que nous ne puisse voir, nos utilisations de ce caractère ne sont limitées que par notre propre imagination. J'avoue que c'est l'un des utilitaires qui me manque le plus lorsque le seul remède pour écrire un texte est un logiciel de traitement de texte.

- { Ce caractère ouvre un groupe. Les groupes sont des blocs de texte auxquels on souhaite appliquer certaines effet ou affecter certaines caractéristiques. Nous en parlerons dans la section ??.
- } Ce caractère clôture un groupe préalablement ouvert avec « { ».
- # Ce caractère est utilisé pour définir les macros. Il fait référence aux arguments de la macro. Voir [section 3.7.1](#) dans ce chapitre.
- _ Introduit un espace blanc insécable dans le document pour éviter un saut de ligne entre les mots qu'il sépare, ce qui signifie que deux mots séparés par le caractère _ resteront toujours sur la même ligne. Nous parlerons de cette instruction et de l'endroit où elle doit être utilisée dans [section 11.3.1](#).
- | Ce caractère est utilisé pour indiquer que deux mots joints par un élément de séparation constituent un mot composé qui peut être divisé par syllabes en la première composante, mais pas en la seconde. Voir section ??.
- \$ Ce caractère est un *interrupteur* pour le mode mathématique. Il active ce mode s'il n'était pas activé, ou le désactive s'il l'était. En mode mathématique, ConTeXt applique des polices et des règles différentes des polices normales, afin d'optimiser l'écriture des formules mathématiques. Bien que l'écriture des mathématiques soit une utilisation très importante de ConTeXt, je ne la développerai pas dans cette introduction. Étant un homme de lettres, je ne me sens pas à la hauteur !
- Ce caractère est utilisé en mode mathématique pour indiquer que ce qui suit doit être mis en indice. Ainsi, par exemple, pour obtenir x_1 , il faut écrire **\$x_1\$**.
- ⁿ Ce caractère est utilisé en mode mathématique pour indiquer que ce qui suit doit être mis en exposant. Ainsi, par exemple, pour obtenir $(x + i)^{n^3}$, il faut écrire **\$x+i)^^{n^3}\$**.

& La documentation de ConTeXt indique qu'il s'agit d'un caractère réservé, mais ne précise pas pourquoi. Ce caractère semble avoir essentiellement deux usages : il est utilisé pour aligner certains éléments verticalement dans les tableaux de base et, dans un contexte mathématique, dans les écritures matricielles . Comme je suis un littéraire, je ne me sens pas capable de faire des tests supplémentaires pour voir à quoi sert précisément ce caractère réservé.

Concernant le choix des caractères réservés, il doit s'agir de caractères disponibles sur la plupart des claviers mais qui ne sont habituellement peu ou pas utilisés dans les écritures. Cependant, bien que peu courants, il est toujours possible que certains d'entre eux apparaissent dans nos documents, comme par exemple lorsque nous voulons écrire que quelque chose coûte 100 dollars (\$100), ou qu'en Espagne, le pourcentage de conducteurs de plus de 65 ans était de 16% en 2018. Dans ces cas, nous ne devons pas écrire le caractère réservé directement, mais utiliser une *commande* qui produira le caractère réservé correctement dans le document final. La commande pour chacun des caractères réservés se trouve dans [table 3.1](#).

Caractère réservé	Commande qui le génère
\	\backslash
%	\%
{	\{
}	\}
#	\#
~	\lettertilde
	\
\$	\\$
-	_
^	\letterhat
&	\&

Tableau 3.1 Ecriture des caractères réservés

Une autre façon d'obtenir les caractères réservés est d'utiliser la commande `\type`. Cette commande envoie ce qu'elle prend comme argument au document final sans le traiter d'aucune manière, et donc sans l'interpréter. Dans le document final, le texte reçu de `\type` sera affiché dans la police monospace typique des terminaux informatiques et des machines à écrire.

Normalement, nous devrions placer le texte que `\type` doit afficher entre accolades. Cependant, lorsque ce texte comprend lui-même des crochets ouvrants ou fermants, nous pouvons, à la place, enfermer le texte entre deux caractères égaux qui ne font pas partie du texte qui constitue l'argument de `\type`. Par exemple : `\type{*...*}`, ou `\type{+...+}`.

Si, par erreur, nous utilisons directement un des caractères réservés autrement que pour l'usage auquel il est destiné, parce que nous avons justement oublié qu'il s'agissait d'un caractère réservé ne pouvant être utilisé comme un caractère normal, trois choses peuvent se produire :

1. Le plus souvent, une erreur est générée lors de la compilation.
2. Nous obtenons un résultat inattendu. Cela se produit surtout avec « ~ » et « % » ; dans le premier cas, au lieu du « ~ » attendu dans le document final, un espace blanc sera inséré ; et dans le second cas, tout ce qui se trouve après « % » sur la même ligne ne sera pas pris en compte par ConTeXt qui le considérera comme commentaire. L'utilisation incorrecte de la « \ » peut également produire un résultat inattendu si elle ou les caractères qui la suivent immédiatement constituent une commande connue de ConTeXt. Cependant, le plus souvent, lorsque nous utilisons incorrectement la « \ », nous obtenons une erreur de compilation.
3. Aucun problème ne se produit : Cela se produit avec trois des caractères réservés utilisés principalement en mathématiques (_ ^ &) : s'ils sont utilisés en dehors de cet environnement, ils sont traités comme des caractères normaux.

Le point 3 est ma conclusion. La vérité est que je n'ai trouvé aucun endroit dans la documentation de ConTeXt qui nous indique où ces caractères réservés peuvent être utilisés directement ; dans mes tests, cependant, je n'ai vu aucune erreur lorsque cela est fait ; contrairement, par exemple, à L^AT_EX.



3.2 Les commandes à proprement parler

⁹ Note:
par convention, pour illustrer quelque chose dans cette introduction, les exemple de code source utilise une police à espace-ment fixe. une coloration syntaxique



de ConTeXt dans un cadre de fond gris.

Le résultat de la compilation est présenté dans un cadre de fond de couleur jaune

Les commandes proprement dites commencent donc toujours par le caractère « \ ». En fonction de ce qui suit immédiatement ce caractère d'échappement, une distinction est faite entre :

- Symboles de contrôle.** Un symbole de contrôle commence par la séquence d'échappement (« \ ») et consiste exclusivement en un caractère autre qu'une lettre, comme par exemple « _ », « \1 », « \' » ou « \% ». Tout caractère ou symbole qui n'est pas une lettre au sens strict du terme peut être un symbole de contrôle, y compris les chiffres, les signes de ponctuation, les symboles et même un espace vide. Dans ce document, pour représenter un espace vide (espace blanc) lorsque sa présence doit être soulignée, le symbole que j'utilise est .. En fait, « _ » (une barre oblique inversée suivie d'un espace blanc) est un symbole de contrôle couramment utilisé, comme nous pourrons bientôt le constater.

Un espace vide ou blanc est un caractère « invisible », ce qui pose un problème dans un document comme celui-ci, où il faut parfois préciser clairement ce qui doit être écrit dans un fichier source. Knuth était déjà conscient de ce problème et, dans son « The TeXBook », il a pris l'habitude de représenter les espaces vides importants par le symbole « .. ». Ainsi, par exemple, si nous voulions montrer que deux mots du fichier source doivent être séparés par deux espaces vides, nous écririons « word1..word2 ».

- Mots de contrôle.** Si le caractère qui suit immédiatement la barre oblique inversée est une lettre à proprement parler, la commande sera un *Mot de contrôle*. Ce groupe de commandes est largement majoritaire. Il a une caractéristique très importante : le nom de la commande ne peut être composé que de lettres ; les chiffres, les signes de ponctuation ou tout autre type de symbole ne sont pas autorisés. Seules les lettres minuscules ou majuscules sont autorisées. N'oubliez pas, par ailleurs, que ConTeXt fait une distinction entre les minuscules et les majuscules, ce qui signifie que les commandes `\mycommand` et `\MyCommand` sont différentes. Mais `\MaCommande1` et `\MaCommande2` seraient considérées comme identiques, puisque n'étant pas des lettres, « 1 » et « 2 » ne font pas partie du nom des commandes.

Le manuel de référence de ConTeXt ne contient aucune règle sur les noms de commande, tout comme le reste des « manuels » inclus avec « ConTeXt Standalone ». Ce que j'ai dit dans le paragraphe précédent est ma conclusion basée sur ce qui se passe dans TeX (où, par ailleurs, des caractères comme les voyelles accentuées qui n'apparaissent pas dans l'alphabet anglais ne sont pas considérés comme des « lettres »).

Lorsque ConTeXt lit un fichier source et trouve le caractère d'échappement (« \ »), il sait qu'une commande va suivre. Il lit alors le premier caractère qui suit la séquence d'échappement. Si ce n'est pas une lettre, cela signifie que la commande est un symbole de contrôle et ne consiste qu'en ce premier symbole. Mais d'un autre côté, si le premier caractère après la séquence d'échappement est une lettre, alors ConTeXt continuera à lire chaque caractère jusqu'à ce qu'il trouve le premier caractère qui ne soit pas une lettre, et il saura alors que le nom de la commande est terminé. C'est pourquoi les noms de commande qui sont des mots de contrôle ne peuvent pas contenir de caractères autres que des lettres.

Lorsque la « non-lettre » à la fin du nom de la commande est un espace vide, il est supposé que l'espace vide ne fait pas partie du texte à traiter, mais qu'il a été inséré exclusivement pour indiquer la fin du nom de la commande, donc ConTeXt se débarrasse de cet espace. Cela produit un effet qui surprend les débutants, car lorsque l'effet de la commande en question implique d'écrire quelque chose dans le document final, la sortie écrite de la commande est liée au mot suivant. Par exemple, les deux phrases suivantes dans le fichier source produisent le résultat suivant :⁹

```
Connaître \TeX aide à l'apprentissage de \ConTeXt.  
Connaître \TeX, si non indispensable, aide à l'apprentissage de \ConTeXt.  
Connaître \TeX      aide à l'apprentissage de \ConTeXt.  
Connaître \TeX{} aide à l'apprentissage de \ConTeXt.  
Connaître \TeX\ aide à l'apprentissage de \ConTeXt.
```

```
Connaître TeXaide à l'apprentissage de ConTeXt.  
Connaître TeX, si non indispensable, aide à l'apprentissage de ConTeXt.  
Connaître TeXaide à l'apprentissage de ConTeXt.  
Connaître TeX aide à l'apprentissage de ConTeXt.  
Connaître TeX aide à l'apprentissage de ConTeXt.
```

Notez comment, dans le premier cas, le mot « TeX » est relié au mot qui suit mais pas dans le second cas. Cela est dû au fait que, dans le premier cas du fichier source, la première « non-lettre » après le nom de la commande \TeX était un espace vide, supprimé parce que ConTeXt a supposé qu'il n'était là que pour indiquer la fin d'un nom de commande, alors que dans le second cas, il y avait une virgule, et comme ce n'est pas un espace vide, il n'a pas été supprimé. Le troisième exemple montre que l'ajout d'espaces blancs supplémentaires ne change rien, car une règle de ConTeXt (que nous verrons dans section 4.2.1) fait qu'un espace blanc « absorbe » tous les blancs et tabulations qui le suivent (1 espace ou 15, c'est pareil).

Par conséquent, lorsque nous rencontrons ce problème (qui heureusement n'arrive pas trop souvent), nous devons nous assurer que la première « non-lettre » après le nom de la commande n'est pas un espace blanc. Il existe deux candidats pour cela :

- Les caractères réservés « {} », utilisé à la quatrième ligne de l'exemple. Le caractère réservé « { », comme je l'ai dit, ouvre un groupe, et « } » ferme un groupe, donc la séquence « {} » introduit un groupe vide. Un groupe vide n'a aucun effet sur le document final, mais il aide ConTeXt à savoir que le nom de la commande qui le précède est terminé. On peut aussi créer un groupe autour de la commande en question, par exemple en écrivant « {\TeX} ». Dans les deux cas, le résultat sera que la première « non-lettre » après \TeX n'est pas un espace vide.

- Le symbole de contrôle « `_` » (une barre oblique inverse suivie d'un espace vide, voir la note sur page 57) utilisé à la cinquième ligne de l'exemple. L'effet de ce symbole de contrôle est d'insérer un espace blanc dans le document final. Pour bien comprendre la logique de ConTeXt, il peut être utile de prendre le temps de voir ce qui se passe lorsque ConTeXt rencontre un mot de contrôle (par exemple `\TeX`) suivi d'un symbole de contrôle (par exemple « `_` ») :
 - ConTeXt rencontre le caractère `\` suivi d'un « `T` » et sachant que cela vient avant un mot de contrôle, il continue à lire les caractères jusqu'à ce qu'il arrive à une « non-lettre », ce qui se produit lorsqu'il arrive au caractère `\` introduisant le prochain symbole de contrôle.
 - Une fois qu'il sait que le nom de la commande est `\TeX`, il exécute la commande et imprime `\TeX` dans le document final. Il retourne ensuite à l'endroit où il a arrêté la lecture pour vérifier le caractère qui suit immédiatement la deuxième barre oblique inversée.
 - Il identifie qu'il s'agit d'un espace vide, c'est-à-dire d'une « non-lettre », ce qui signifie qu'il s'agit d'un symbole de contrôle, qu'il peut donc exécuter. Il le fait et insère un espace vide.
 - Enfin, il revient une fois de plus au point où il a arrêté la lecture (l'espace blanc qui était le symbole de contrôle) et continue à traiter le fichier source à partir de là.

J'ai expliqué ce mécanisme de manière assez détaillée, car l'élimination des espaces vides surprend souvent les nouveaux venus. Il convient toutefois de noter que le problème est relativement mineur, car les mots de contrôle ne s'impriment généralement pas directement dans le document final, mais en affectent le format et l'apparence. En revanche, il est assez fréquent que les symboles de contrôle s'impriment sur le document final.

Il existe une troisième procédure pour éviter le problème des espaces vides, qui consiste à définir (à la manière de `\TeX`) une commande similaire et à inclure une « non-lettre » à la fin du nom de la commande. Par exemple, la séquence suivante :

```
\def\txt-{\TeX}
```

créerait une commande appelée `\txt`, qui aurait exactement la même fonction que la commande `\TeX` et ne fonctionnerait correctement que si elle était suivie d'un trait d'union `\txt-`. Ce trait d'union ne fait pas techniquement partie du nom de la commande, mais celle-ci ne fonctionnera que si le nom est suivi d'un trait d'union. La raison de cette situation est liée au mécanisme de définition des macros `\TeX` et est trop complexe pour être expliquée ici. Mais cela fonctionne : une fois cette commande définie, chaque fois que nous utilisons `\txt-`, ConTeXt la remplace par `\TeX` en éliminant le trait d'union, mais en l'utilisant en interne pour savoir que le nom de la commande est déjà terminé, de sorte qu'un espace blanc immédiatement après ne serait pas supprimé.

Cette « astuce » ne fonctionnera pas correctement avec la commande `\define`, qui est une commande spécifiquement ConTeXt pour définir des macros.

3.3 Périmètre des commandes

3.3.1 Les commandes qui nécessite ou pas une périmètre d'application

De nombreuses commandes ConTeXt en particulier celles qui affectent les fonctions de formatage des polices (gras, italique, petites capitales, etc.), activent une certaine fonction qui reste activée jusqu'à ce qu'une autre commande la désactive ou active une autre fonction incompatible avec elle. Par exemple, la commande `\bf` active le gras, et elle restera active jusqu'à ce qu'elle trouve une commande *incompatible* comme, par exemple, `\tf`, ou `\it`.

Ces types de commandes n'ont pas besoin de prendre d'argument, car elles ne sont pas conçues pour s'appliquer uniquement à certains textes. C'est comme si elles se limitaient à *activer* une fonction quelconque (gras, italique, sans serif, taille de police donnée, etc.).

Lorsque ces commandes sont exécutées dans un *groupe* (voir section 3.8.1), elles perdent également leur efficacité lorsque le groupe dans lequel elles sont exécutées est fermé. Par conséquent, pour que ces commandes n'affectent qu'une partie du texte, il faut souvent générer un groupe contenant cette commande et le texte que l'on souhaite qu'elle affecte. Un groupe est créé en l'enfermant entre des accolades. Par conséquent, le texte suivant

```
In {\it The \TeX Book}, {\sc Knuth} explained \bf{everything} you need to know about \TeX.
```

In *The T_EXBook*, KNUTH explained **everything you need to know about Te_X**.

crée deux groupes, l'un pour déterminer la portée de la commande `\it` (italique) et l'autre pour déterminer la portée de la commande `\sc` (petites capitales, small capital en anglais).

Au contraire de ce type de commande, il en existe d'autres qui nécessitent immédiatement une indication du texte auquel elles doivent être appliquées. Dans ce cas, le texte qui doit être affecté par la commande est placé entre des crochets immédiatement après la commande. Par exemple, nous pouvons citer la commande `\framed` : cette commande dessine un cadre autour du texte qu'elle prend comme argument, par exemple :

```
\framed{Tweedledum and Tweedledee}
```

Tweedledum and Tweedledee

¹⁰ Pas toujours, cela dépend de l'environnement en question et de la situation dans le reste du document. ConTeXt diffère de L^AT_EX à cet égard qui est beaucoup plus stricte.

¹¹ test

Notez que, bien que dans le premier groupe de commandes (celles qui ne requièrent pas d'argument), les accolades sont parfois utilisées pour déterminer le champ d'action, mais cela n'est pas nécessaire pour que la commande fonctionne. La commande est conçue pour être appliquée à partir du point où elle apparaît. Ainsi, lorsque vous déterminez son champ d'application en utilisant des crochets, la commande est placée *entre ces crochets*, contrairement au deuxième groupe de commandes, où les parenthèses encadrant le texte auquel la commande doit être s'appliquer, sont placés après le commandement.

Dans le cas de la commande `\framed`, il est évident que l'effet qu'elle produit nécessite un argument – le texte auquel elle est appliquée. Dans d'autres cas, cela dépend du programmeur si la commande est d'un type ou d'un autre. Ainsi, par exemple, les commandes `\it` et `\color` sont assez similaires : elles appliquent une caractéristique (format ou couleur) au texte. Mais la décision a été prise de programmer la première sans argument, et la seconde comme une commande avec un argument.

3.3.2 Commandes nécessitant d'indiquer leur début et fin d'application (environnements)

Certaines commandes fonctionnent par couple afin de déterminer leur portée, en indiquant précisément le moment où elles commencent à être appliquées et celui où elles cessent de l'être. Ces commandes sont donc présentées par paires : l'une indique le moment où la commande doit être activée, et l'autre celui où cette action doit cesser. La commande « `start` », suivie du nom de la commande, est utilisée pour indiquer le début de l'action, et la commande « `stop` », également suivie du nom de la commande, pour indiquer la fin. Ainsi, par exemple, la commande « `itemize` » devient `\startitemize` pour indiquer le début d'une *liste d'items* et `\stopitemize` pour indiquer la fin.

Il n'y a pas de nom spécial pour ces paires de commandes dans la documentation officielle de ConTeXt. Le manuel de référence et l'introduction les appellent simplement « `start ... stop` ». Parfois elles sont appelés *environnements*, qui est également le nom que L^AT_EX donne à un type de construction similaire, mais cela présente un inconvénient dans ConTeXt car ce terme « *environnement* » est également utilisé pour autre chose (un type spécial de fichier source que nous verrons lorsque nous parlerons des projets multifichiers dans section ??). Néanmoins, puisque le terme environnement est clair, et que le contexte permettra de distinguer facilement si nous parlons de *commandes d'environnement* ou de *fichiers d'environnement*, j'utiliserai ce terme.

Les environnements consistent donc en une commande qui les ouvre, les commence, et une autre qui les ferme, les termine. Si le fichier source contient une commande d'ouverture d'environnement qui n'est pas fermée par la suite, une erreur est normalement générée.¹⁰ D'autre part, ces types d'erreurs sont plus difficiles à trouver, car l'erreur peut se produire bien au-delà de l'endroit où se trouve la commande d'ouverture. Parfois, le fichier « `.log` » nous montrera la ligne où commence

l'environnement incorrectement fermé ; mais d'autres fois, l'absence d'une fermeture¹¹ d'environnement va impliquer une mauvaise interprétation par ConTeXt qui soulignera le passage qu'il considère comme éronné et non pas le manque de fermeture d'environnement, ce qui signifie que le fichier « .log » ne nous est pas d'une grande aide pour trouver où se situe le problème.

Les environnements peuvent être imbriqués, ce qui signifie qu'un autre environnement peut être ouvert à l'intérieur d'un environnement existant. Dans de tels cas un environnement doit absolument être fermé à l'intérieur de l'environnement dans lequel il a été ouvert. En d'autres termes, l'ordre dans lequel les environnements sont fermés doit être cohérent avec l'ordre dans lequel ils ont été ouverts. Je pense que cela devrait être clair à partir de l'exemple suivant :

```
\startQuelqueChose
  ...
    \startQuelqueChoseAutre
      ...
        \startEncoreQuelqueChoseAutre
          ...
            \stopEncoreQuelqueChoseAutre
        \stopQuelqueChoseAutre
      \stopQuelqueChose
```

Dans l'exemple, vous pouvez voir comment l'environnement « QuelqueChoseAutre » a été ouvert à l'intérieur de l'environnement « QuelqueChose » et doit être fermé à l'intérieur de celui-ci également. Dans le cas contraire, une erreur se produirait lors de la compilation du fichier.

En général, les commandes conçues comme *environnements* sont celles qui mettent en œuvre un changement destiné à être appliqué à des unités de texte au moins aussi grande que le paragraphe. Par exemple, l'environnement « narrower » qui modifie les marges, n'a de sens que lorsqu'il est appliqué au niveau du paragraphe, ou l'environnement « framedtext » qui encadre un ou plusieurs paragraphes. Ce dernier environnement peut nous aider à comprendre pourquoi certaines commandes sont conçues comme des environnements et d'autres comme des commandes individuelles : si nous souhaitons encadrer un ou plusieurs mots, tous sur la même ligne, nous utiliserons la commande `\framed`, mais si ce que nous voulons encadrer est un paragraphe entier (ou plusieurs paragraphes), nous utiliserons l'environnement « `\startframed` » ou « `\startframedtext` ».

D'autre part, le texte situé dans un environnement particulier constitue normalement un *groupe* (voir section ??), ce qui signifie que si une commande d'activation est trouvée à l'intérieur d'un environnement, parmi les commandes qui s'appliquent à tout le texte qui suit, cette commande ne s'appliquera que jusqu'à la fin de l'environnement dans lequel elle se trouve. En fait, ConTeXt a un *environnement sans nom* commençant par la commande `\start` (aucun autre texte ne suit ; juste `start`, c'est pourquoi je l'appelle un *environnement sans nom*) et se termine par la commande `\stop`. Je pense que la seule fonction de cette commande est de créer un groupe.



Je n'ai lu nulle part dans la documentation de ConTeXt que l'un des effets des environnements est de grouper leur contenu, mais c'est le résultat de mes tests avec un certain nombre d'environnements prédefinis, bien que je doive admettre que mes tests n'ont pas été trop exhaustifs. J'ai simplement vérifié quelques environnements choisis au hasard. Mes tests montrent cependant qu'une telle affirmation, si elle était vraie, ne le serait que pour certains environnements prédefinis : ceux créés avec la commande `\definestartstop` (expliquée dans la section 3.7.2) ne créent aucun groupe, à moins que lors de la définition du nouvel environnement nous n'inclions les commandes nécessaires à la création du groupe (voir section 3.8.1).

Je suppose également que l'environnement que j'ai appelé le *sans nom* (`\start`) n'est là que pour créer un groupe : il crée effectivement un groupe, mais je ne sais pas s'il a ou non une autre utilité. C'est l'une des commandes non documentées du manuel de référence.

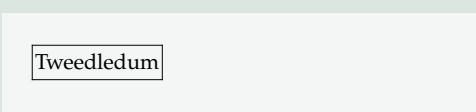
3.4 Options de fonctionnement des commandes

3.4.1 Commandes qui peuvent fonctionner de différentes façon distinctes

De nombreuses commandes peuvent fonctionner de plusieurs façons. Dans ce cas, il existe toujours une manière prédéterminée de travailler (une manière par défaut) qui peut être modifiée en indiquant les paramètres correspondant à l'opération souhaitée entre crochets après le nom de la commande.

Un bon exemple est la commande `\framed` mentionnée dans la section précédente. Cette commande dessine un cadre autour du texte qu'elle prend comme argument. Par défaut, le cadre a la hauteur et la largeur du texte auquel il est appliqué, mais nous pouvons indiquer une hauteur et une largeur différentes. Ainsi, nous pouvons voir la différence entre le fonctionnement de la commande `\framed` par défaut :

```
\framed{Tweedledum}
```



Tweedledum

et celui d'une version personnalisée :

```
\framed  
[width=3cm, height=1cm]  
{Tweedledum}
```



Tweedledum

Dans le deuxième exemple, nous avons indiqué entre les crochets une largeur et une hauteur spécifiques pour le cadre qui entoure le texte qu'il prend comme argument. À l'intérieur des crochets, les différentes options de configuration sont séparées par une virgule ; les espaces vides et même les sauts de ligne (à condition qu'il ne s'agisse pas d'un double saut de ligne) entre deux ou plusieurs options, ne sont pas pris en considération afin que, par exemple, les quatre versions suivantes de la même commande produisent exactement le même résultat :

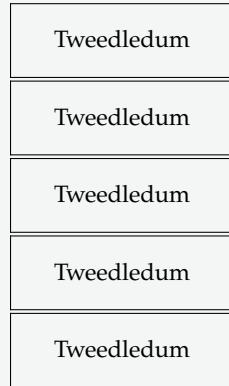
```
\framed[width=3cm,height=1cm]{Tweedledum}

\framed[width=3cm, height=1cm]{Tweedledum}

\framed
[width=3cm, height=1cm]
{Tweedledum}

\framed
[width=3cm,
 height=1cm]
{Tweedledum}

\framed
[
 width=3cm,
 height=1cm,
]
{Tweedledum}
```



Il est évident que la version finale est la plus facile à lire : nous pouvons voir du premier coup d'œil combien d'options utilisées et à quelle contenu s'applique la commande. Dans un exemple comme celui-ci, avec seulement deux options, cela ne semble peut-être pas si important ; mais dans les cas où il y a une longue liste d'options, si chacune d'entre elles a sa propre ligne dans le fichier source, il est plus facile de *comprendre* ce que le fichier source demande à ConTeXt de faire, et aussi, si nécessaire, de découvrir une erreur potentielle (car il est possible de commenter successivement chaque ligne et donc chaque option). Par conséquent, ce dernier format (ou un format similaire) pour l'écriture des commandes est celui qui est « préféré et conseillé » par les utilisateurs.

Quant à la syntaxe des options de configuration, voir plus loin dans ([section 3.5](#)).

3.4.2 Les commandes qui configurent comment d'autres commandes fonctionnent (`\setupQuelque-Chose`)

Nous avons déjà vu que les commandes qui offrent des options de fonctionnement ont toujours un jeu d'options par défaut. Si l'une de ces commandes est appelée plusieurs fois dans notre fichier source, et que nous souhaitons modifier la valeur par défaut pour toutes ces commandes, plutôt que de modifier ces options à chaque fois que la commande est appelée, il est beaucoup plus pratique et efficace de modifier la valeur par défaut. Pour ce faire, il existe presque toujours une commande dont le nom commence par `\setup`, suivi du nom de la commande dont nous souhaitons modifier les options par défaut.

La commande `\framed` que nous avons utilisée comme exemple dans cette section reste un bon exemple. Ainsi, si nous utilisons beaucoup de cadres dans notre document, mais qu'ils nécessitent tous des mesures précises, il serait préférable de reconfigurer le fonctionnement de `\framed`, en le faisant avec `\setupframed`. Ainsi,

```
\setupframed
[
  width=3cm,
  height=1cm,
]
```

fera en sorte qu'à partir de cette déclaration dans le code source, chaque fois que nous appellerons `\framed`, il générera par défaut un cadre de 3 centimètres de large sur 1 centimètre de haut, sans qu'il soit nécessaire de l'indiquer expressément à chaque fois. Dans le vocabulaire des logiciels de traitement de texte, cela peut être rapproché de la définition d'un élément de style.

Il existe environ 300 commandes dans ConTeXt qui nous permettent de configurer le fonctionnement d'autres commandes. Ainsi, nous pouvons configurer le fonctionnement par défaut de (`\framed`), des listes (« `itemize` »), des titres de chapitre (`\chapter`), ou des titres de section (`\section`), etc.

3.4.3 Définir des versions personnalisée de commande configurables (`\defineQuelqueChose`)

En continuant avec l'exemple du `\framed`, il est évident que si notre document utilise plusieurs types de cadres, chacun avec des mesures différentes, l'idéal serait de pouvoir *prédefinir* différentes configurations de `\framed`, et de les associer à un nom particulier afin de pouvoir utiliser l'un ou l'autre selon les besoins. Nous pouvons le faire dans ConTeXt avec la commande `\defineframed`, dont la syntaxe est :

```
\defineframed
[MonCadre]
[MaConfigurationPourCadre]
```

où *MonCadre* est le nom attribué au type particulier de cadre à configurer ; et *MaConfigurationPourCadre* est la configuration particulière associée à ce nom.

L'association entre la configuration et le nom se traduit par l'existence d'une nouvelle fonction « *MonCadre* » que nous pourrons l'utiliser dans n'importe quel contexte où nous aurions pu utiliser la commande originale (`\framed`).

Cette possibilité n'existe pas seulement pour le cas concret de la commande `\framed`, mais pour de nombreuses autres commandes. La combinaison de `\defineQuelqueChose + \setupQuelqueChose` est un mécanisme qui donne à ConTeXt son extrême puissance et flexibilité. Si nous examinons en détail ce que fait la commande `\defineSomething`, nous constatons que :

- Tout d'abord, elle clone une commande particulière qui supporte toute une série d'option et de configurations. Par cette opération, le clone *hérite* de la commande initiale et de sa configuration par défaut.
- Il associe ce clone au nom d'une nouvelle commande.
- Enfin, il définit une configuration prédéterminée pour le clone, différente de celle de la commande originale.

Dans l'exemple que nous avons donné, nous avons configuré notre cadre spécial « MonCadre » en même temps que nous le créions. Mais nous pouvons aussi le créer d'abord et le configurer ensuite, car, comme je l'ai dit, une fois le clone créé, il peut être utilisé là où l'original aurait pu l'être. Ainsi, dans notre exemple, nous pouvons le configurer avec `\setupframed` en indiquant le nom du cadre (framed) que nous voulons configurer. Dans ce cas, la commande `\setup` prendra un nouvel argument avec le nom du cadre à configurer :

```
\defineframed  
  [MonCadre]  
  
\setupframed  
  [MonCadre]  
  [MaConfigurationPourCadre]
```

3.5 Résumé sur la syntaxe des commandes et des options, et sur l'utilisation des crochets et des accolades lors de leur appel.

this section is especially dedicated to LaTeX users, so they can understand the different use of such brackets.

En résumant ce que nous avons vu jusqu'à présent, nous voyons que dans ConTeXt

- Les commandes commencent toujours par le caractère « \ ».
- Certaines commandes peuvent prendre un ou plusieurs arguments.
- Les arguments qui indiquent à la commande *comment* elle doit fonctionner ou qui affectent son fonctionnement d'une manière ou d'une autre, sont introduits entre crochets.
- Les arguments qui indiquent à la commande sur quelle partie du texte elle doit agir sont présentés entre accolades.
Lorsque la commande ne doit agir que sur une seule lettre, comme c'est le cas, par exemple, de la commande `\buildtextcedilla` (pour donner un exemple – le « ç » si souvent utilisée en catalan), les accolades autour de l'argument peuvent être omis : la commande s'appliquera au premier caractère qui n'est pas un espace blanc.
- Certains arguments sont facultatifs, auquel cas nous pouvons les omettre. Mais ce que nous ne pouvons jamais faire, c'est changer l'ordre des arguments que la commande attend.

Les arguments introduits entre crochets peuvent être de différent type : un nom symbolique (dont ConTeXt connaît la signification), une mesure ou une dimension, un nombre, le nom d'une autre commande.

Ils peuvent prendre trois forme différentes ::

- une information unique
- une série d'informations uniques, séparées par des virgules
- une série d'informations sous la forme de couple « clé=valeur », utilisant pour clé des noms de variables auxquelles il faut donner une valeur. Dans ce cas, la définition officielle de la commande (voir [section 3.6](#)) fournit un guide utile pour connaître les clés disponibles et le type de valeur attendu pour chacune.

Enfin, il n'arrive jamais avec ConTeXt qu'au sein d'un même argument on mélange le format de déclaration. Nous pouvons donc avoir les cas de figures suivants

```
\commande[Option1, Option2, ...]  
\commande[Variable1=valeur, Variable2=valeur, ...]  
\commande[Option1] [Variable1=valeur, Variable2=valeur, ...]
```

Mais nous n'aurons jamais un mélange du genre :

```
\commande[Option1, Variable1=valeur, ...]
```

Certaines règles syntaxiques sont à bien prendre en compte :

- Les espaces et les sauts de ligne entre les différents arguments d'une commande sont ignorés.
- une information utilisée dans l'argument peut contenir des espaces vides ou des commandes. Dans ce cas, il est fortement conseillée de la placer entre accolades.
- Les espaces et les sauts de ligne (autres que les doubles) entre les différentes informations sont ignorés.
- Par contre, et ceci est une erreur très commune, entre la première lettre de la clé et la virgule indiquant la fin du couple « clé=valeur », les espaces ne sont pas ignorés. Les règles syntaxiques consistent donc à juxtaposer sans aucun espace le mot clé, le signe égal, la valeur et la virgule. Pour prendre en compte des espaces dans la valeur, la pratique est encore une fois de la mettre entre accolades.
- Nous devons également inclure le contenu de la valeur entre accolades si elle intègre elle-même des crochets. Sinon le premier crochet fermant sera considéré comme fermant non seulement la valeur mais aussi l'argument que nous sommes en train de définir. Voyez :

```
\startsection[title=mon titre[5] avec crochets]
  Du texte pour cette section
  NE FONCTIONNERA PAS
\stopsection
\startsection[title={mon titre[5] avec
crochets}]
  Du texte pour cette section
  FONCTIONNERA
\stopsection
```

1 mon titre[5]

avec crochets] Du texte pour cette sect
NERA PAS

2 mon titre[5] avec crochets

Du texte pour cette section FONCTIO

3.6 La liste officielle des commandes ConTeXt

Parmi la documentation de ConTeXt il existe un document particulièrement important contenant la liste de toutes les commandes, et indiquant pour chacune d'entre elles combien d'arguments elles attendent et de quel type, ainsi que les différentes options possibles et leurs valeurs autorisées. Ce document s'appelle « `setup-en.pdf` », et est généré automatiquement pour chaque nouvelle version de ConTeXt. Il se trouve dans le répertoire appelé « `tex/texmf-context/doc/context/documents/general/qrcs` ».

En fait, la « `qrc` » possède sept versions de ce document, une pour chacune des langues disposant d'une interface ConTeXt : allemand, tchèque, français, néerlandais, anglais, italien et roumain. Pour chacune de ces langues, il existe deux documents dans le répertoire : un appelé « `setup-LangCode` » (où `LangCode` est le code en deux lettres d'identification des langues internationales) et un second document appelé « `setup-mapping-LangCode` ». Ce second document contient une liste de commandes par ordre alphabétique et indique la commande `prototype`, mais sans les informations des valeurs possibles pour chaque argument.

Ce document est fondamental pour apprendre à utiliser ConTeXt, car c'est là que nous pouvons savoir si une certaine commande existe ou non ; ceci est particulièrement utile, compte tenu de la combinaison `COMMANDÉ` (ou `ENVIRONNEMENT`) + `setup-COMMANDÉ` + `defineCOMMANDÉ`. Par exemple, si je sais qu'une ligne vierge est introduite avec la commande `\blank`, je peux savoir s'il existe une commande appelée `\setupblank` qui me permet de la configurer, et une autre qui me permet d'établir une configuration personnalisée pour les lignes vierges, (`\defineblank`).

« `setup-fr.pdf` » est donc fondamental pour l'apprentissage de ConTeXt. Mais je préférerais vraiment, tout d'abord, qu'il nous dise si une commande ne fonctionne que dans Mark II ou Mark IV, et surtout, qu'au lieu de nous indiquer seulement la liste et le type d'arguments que chaque commande autorise, il nous dise à quoi servent ces arguments. Cela réduirait considérablement les lacunes de la documentation de la ConTeXt. Certaines commandes autorisent des arguments facultatifs que je ne mentionne même pas dans cette introduction parce que je ne sais pas à quoi ils servent et, puisqu'ils sont facultatifs, il n'est pas nécessaire de les mentionner. C'est extrêmement frustrant.

La méthode mise en oeuvre par la communauté ConTeXt est dorénavant de documenter tout cela dans le [Wiki](#) avec une adresse web spécifique pour chaque commande, par exemple pour `\setupframed` : <https://wiki.contextgarden.net/index.php?title=Command/setupframed>

Ainsi, chaque utilisateur est invité à compléter progressivement la documentation au fil de ses découvertes, souvent issues des échanges sur [la liste de diffusions NTG-context](#) où les développeurs demanderont à Wikifier les réponses apportées.

3.7 Définir de nouvelles commandes

3.7.1 Mécanisme général pour définir de nouvelles commandes

Nous venons de voir comment, avec `\defineQuelqueChose`, nous pouvons cloner une commande préexistante et développer une nouvelle version de celle-ci qui à partir de là, fonctionnera comme une nouvelle commande.

En plus de cette possibilité, qui n'est disponible que pour certaines commandes spécifiques (quelques-unes, certes, mais pas toutes), ConTeXt a un mécanisme général pour définir de nouvelles commandes qui est extrêmement puissant mais aussi, dans certaines de ses utilisations, assez complexe. Dans un texte comme celui-ci, destiné aux débutants, je pense qu'il est préférable de le présenter en commençant par certaines de ses utilisations les plus simples. La plus simple de toutes est d'associer des bouts de texte à un mot, de sorte que chaque fois que ce mot apparaît dans le fichier source, il est remplacé par le texte qui lui est lié. Cela nous permettra, d'une part, d'économiser beaucoup de temps de frappe et, d'autre part, comme avantage supplémentaire, de réduire les possibilités de faire des erreurs de frappe, tout en s'assurant que le texte en question est toujours écrit de la même façon.

Imaginons, par exemple, que nous sommes en train d'écrire un traité sur l'allitération dans les textes latins, où nous citons souvent la phrase latine « *O Tite tute Tati, tibi tanta, tyranne, tulisti !* ». (C'est toi-même, Titus Tatius, qui t'es fait, à toi, tyran, tant de torts !). Il s'agit d'une phrase assez longue dont deux des mots sont des noms propres et commencent par une majuscule, et où, avouons-le, même si nous aimons la poésie latine, il nous est facile de « trébucher » en l'écrivant. Dans ce cas, nous pourrions simplement mettre dans le préambule de notre fichier source :

```
\define\Tite[\quotation{O Tite tute Tati, tibi tanta, tyranne, tulisti}]
```

Sur la base d'une telle définition, chaque fois que la commande `\Tite` apparaîtra dans notre fichier source, elle sera remplacée par la séquence indiquée : la phrase elle-même, prise comme argument de la commande `\quotation` qui met son argument entre guillemets en respectant les règles typographiques de la langue du document. Cela nous permet de garantir que la façon dont cette phrase apparaîtra sera toujours la même. Nous aurions également pu l'écrire en italique, avec une taille de police plus grande... comme bon nous semble. L'important, c'est que nous ne devons l'écrire qu'une seule fois et qu'elle sera reproduite dans tout le texte exactement comme elle a été écrite, aussi souvent que nous le voulons. Nous pourrions également créer deux versions de la commande, appelées `\Tite` et `\tite`, selon que la phrase doit être écrite en majuscules ou non. De plus, il suffira de modifier la définition et elle sera répercutee automatiquement dans l'ensemble du document.

Le texte de remplacement peut être du texte pur, ou inclure des commandes, ou encore former des expressions mathématiques dans lesquelles il y a plus de chances de faire des fautes de frappe (du moins pour moi). Par exemple, si l'expression (x_1, \dots, x_n) doit apparaître régulièrement dans notre texte, nous pouvons créer une commande pour la représenter. Par exemple

```
\startTEXpage %
\environment introCTX_env_09_for_demo %
\setupbodyfont[palatino,9pt] %
\framed[align=normal,
width=\dimexpr\textwidth-\marged\relax,
offset=6pt,
frame=off,strut=no]{%
\define\Tite{\quotation{0 Tite tute Tati, tibi tanta, tyranne, tulisti}}}
```

de sorte que chaque fois que `\xvec` apparaît dans le code source, il sera remplacé par l'expression qui lui est associée durant la compilation par ConTeXt.

La syntaxe générale de la commande `\define` est la suivante :

```
\define[NbrArguments]{NomCommande}{TexteOuCodeDeSubstitution}
```

où

- **NbreArguments** désigne le nombre d'arguments que la nouvelle commande prendra. Si elle n'a pas besoin d'en prendre, comme dans les exemples donnés jusqu'à présent, elle est omise.
- **NomCommande** désigne le nom que portera la nouvelle commande. Les règles générales relatives aux noms de commande s'appliquent ici. Le nom peut être un caractère unique qui n'est pas une lettre, ou une ou plusieurs lettres sans inclure de caractère « non-lettre ».
- **TexteOuCodeDeSubstitution** contient le texte ou le code source qui sera substitué à la commande à chacune des ses occurrences dans le fichier source.

La possibilité de fournir aux nouvelles commandes des arguments dans leur définition confère à ce mécanisme une grande souplesse, car elle permet de définir un texte de remplacement variable en fonction des arguments pris.

Par exemple : imaginons que nous voulions écrire une commande qui produise l'ouverture d'une lettre commerciale. Une version très simple de cette commande serait la suivante

```
\define\EnTetedeLettre{
  \rightaligned{Anne Smith}\par
  \rightaligned{Consultant}\par
  Marseille, \date\par
  Chère Madame,\par}
\EnTetedeLettre
```

Marseille, June 27, 2021
Chère Madame,

Anne Smith
Consultant

mais il serait préférable d'avoir une version de la commande qui écrirait le nom du destinataire dans l'en-tête. Cela nécessiterait l'utilisation d'un paramètre qui communiquerait le nom du destinataire à la nouvelle commande. Il faudrait donc redéfinir la commande comme suit :

```
\define[1]\EnTetedeLettre{  
    \rightaligned{Anne Smith}\par  
    \rightaligned{Consultant}\par  
    Marseille, \date\par  
    Chère Madame #1,\par}  
\EnTetedeLettre{Dupond}
```

Marseille, June 27, 2021
Chère Madame Dupond,

Anne Smith
Consultant

Notez que nous avons introduit deux changements dans la définition. Tout d'abord, entre le mot clé `\define` et le nouveau nom de la commande, nous avons inclus un 1 entre crochets ([1]). Cela indique à ConTeXt que la commande que nous définissons prendra un argument.

Plus loin, à la dernière ligne de la définition de la commande, nous avons écrit « Chère Madame #1 », en utilisant le caractère réservé « # ». Cela indique qu'à l'emplacement où apparaît « #1 », le contenu du premier argument sera inséré.

Si elle avait deux paramètres, « #1 » ferait référence au premier paramètre et « #2 » au second. Afin d'appeler la commande (dans le fichier source) après le nom de la commande, les arguments doivent être inclus entre accolades, chaque argument ayant son propre ensemble. Ainsi, la commande que nous venons de définir doit être appelée de la manière suivante : « `\EnTetedeLettre{Nom du destinataire}` », tel que cela est fait dans l'exemple.

Nous pourrions encore améliorer la fonction précédente, car elle suppose que la lettre sera envoyée à une femme (elle met « chère Madame »), alors que nous pourrions peut-être inclure un autre paramètre pour distinguer les destinataires masculins et féminins. par exemple :

```
\define[2]\EnTetedeLettre{  
    \rightaligned{Anne Smith}\par  
    \rightaligned{Consultant}\par  
    Marseille, \date\par  
    #1\ #2,\par}  
\EnTetedeLettre{Cher Monsieur}{Antoine  
Dupond}
```

Marseille, June 27, 2021
Cher Monsieur Antoine Dupond,

Anne Smith
Consultant

bien que cela ne soit pas très élégant (du point de vue de la programmation). Il serait préférable que des valeurs symboliques soient définies pour le premier argument (homme/femme ; 0/1 ; m/f) afin que la macro elle-même choisisse le texte approprié en fonction de cette valeur. Mais pour expliquer comment y parvenir, il faut aller plus en profondeur que ce que je pense que le lecteur novice peut comprendre à ce stade.

3.7.2 Cration de nouveaux environnements

Pour crer un nouvel environnement, ConTeXt fournit la commande `\definesstartstop` dont la syntaxe est la suivante :

```
\definesstartstop[Nom] [Configuration]
```

Dans la definition *officielle* de `\definesstartstop` (voir section 3.6) il y a un argument supplementaire que je n'ai pas mis ci-dessus parce qu'il est optionnel, et je n'ai pas te capable de trouver a quoi il sert . Ni le manuel d'introduction « [ConTeXt Mark IV, an Excursion](#) », ni le manuel de reference ne l'expliquent. J'avais suppos que cet argument (qui doit tre saisi entre le nom et la configuration) pouvait tre le nom d'un environnement existant qui servirait de modle initial pour le nouvel environnement, mais mes tests montrent que cette hypothese tait fausse. J'ai consult la liste de diffusion ConTeXt et je n'ai vu aucune utilisation de cet argument possible.



o

- **Nom** est le nom que portera le nouvel environnement.
- **Configuration** nous permet de configurer le comportement du nouvel environnement. Nous disposons des valeurs suivantes avec lesquelles nous pouvons le configurer :
 - `before` : Commandes a executer avant d'entrer dans l'environnement.
 - `after` : Commandes a executer aprs avoir quitt l'environnement.
 - `style` : Style que doit avoir le texte du nouvel environnement.
 - `setups` : Ensemble de commandes cres avec `\startsetups ... \stopsetups`. Cette commande et son utilisation ne sont pas expliques dans cette introduction.
 - `color` : Couleur a appliquer au texte
 - `inbetween, left, right` : Options non documentes que je n'ai pas russi  faire fonctionner . D'aprs les tests que j'ai effectus, indiquant une certaine valeur pour ces options, je ne vois aucun changement dans l'environnement. Il est possible que l'impact ne concerne pas l'environnement mais la commande qui semble crer avec `\Nom`. Une piste sur [la liste de diffusions NTG-context](#).



Un exemple de la definition d'un environnement pourrait tre le suivant :

```

\definemystop
[TextWithBar]
[before=\bgroup\startmarginrule\noindentation,
 after=\stopmarginrule\egroup,
 style=\ss,
 color=darkyellow]

\starttext
The first two fundamental laws of human stupidity state unambiguously
that:
\startTextWithBar
\startitemize[n,broad]
\item Always and inevitably we underestimate the number of stupid
individuals in the world.
\item The probability that a given person is stupid is independent
of any other characteristic of the same person.
\stopitemize
\stopTextWithBar
\stoptext

```

The first two fundamental laws of human stupidity state unambiguously that:

1. Always and inevitably we underestimate the number of stupid individuals in the world.
2. The probability that a given person is stupid is independent of any other characteristic of the same person.

Si nous voulons que notre nouvel environnement soit un groupe ([section 3.8.1](#)), de sorte que toute altération du fonctionnement normal de ConTEXt qui se produit en son sein disparaîsse en quittant l'environnement, nous devons inclure la commande `\bgroup` dans l'option « `before` », et la commande `\egroup` dans l'option « `after` ».

3.8 Autres concepts fondamentaux

Il existe d'autres notions, autres que les commandes, qui sont fondamentales pour comprendre la logique du fonctionnement de ConTeXt. Certaines d'entre elles, en raison de leur complexité, ne sont pas appropriées pour une introduction et ne seront donc pas abordées dans ce document ; mais il y a deux notions qu'il convient d'examiner maintenant : les groupes et les dimensions.

¹² La notion de *boîte* est également une notion centrale de ConTeXt mais son explication n'est pas incluse dans cette introduction. Vous pouvez voir le manuel dédié

3.8.1 Groupes

Un groupe est un fragment bien défini du fichier source que ConTeXt utilisé comme une *unité de travail*. (ce que cela signifie est expliqué plus loin). Chaque groupe a un début et une fin qui doivent être expressément indiqués. Un groupe commence :

- Avec le caractère réservé « { » ou avec la commande `\bgroup`.
- Avec la commande `\begingroup`.
- Avec la commande `\start`
- Avec l'ouverture de certains environnements (commande `\startSomething`).
- En commençant un environnement mathématique (avec le caractère réservé "\$").

et est fermé

- Avec le caractère réservé « } » ou avec la commande `\egroup`.
- Avec la commande `\endgroup`.
- Avec la commande `\stop`
- Avec la fermeture de l'environnement (commande `\stopSomething`).
- Lors de la sortie de l'environnement mathématique (avec le caractère réservé "\$").

Certaines commandes génèrent aussi automatiquement un groupe, par exemple, `\hbox`, `\vbox` et, en général, les commandes liées à la création de *boîte*¹². En dehors de ces derniers cas (groupes générés automatiquement par certaines commandes), la manière de fermer un groupe doit être cohérente avec la manière dont il a été ouvert. Cela signifie qu'un groupe commencé avec « { » doit être fermé avec « } », et qu'un groupe commencé avec `\begingroup` doit être fermé avec `\endgroup`. Cette règle n'a qu'une seule exception : un groupe commencé par « { » peut être fermé par `\egroup`, et le groupe commencé par `\bgroup` peut être fermé par « } » ; en réalité, cela signifie que « { » et `\bgroup` sont complètement synonymes et interchangeables, et de même pour « } » et `\egroup`.

Les commandes `\bgroup` et `\egroup` ont été conçues pour pouvoir définir des commandes pour ouvrir un groupe et d'autres pour fermer un groupe. Par conséquent, pour des raisons internes à la syntaxe TeX ces groupes ne pouvaient pas être ouverts et fermés avec des accolades, car cela aurait généré des accolades déséquilibrées dans le fichier source, ce qui aurait toujours provoqué une erreur lors de la compilation.

En revanche, les commandes `\begingroup` et `\endgroup` ne sont pas interchangeables avec les accolades ou les commandes `\bgroup ... \egroup`, car un groupe commencé avec `\begingroup` doit être fermé avec `\endgroup`. Ces dernières commandes ont été conçues pour permettre une vérification beaucoup plus approfondie des erreurs. En général, les utilisateurs normaux n'ont pas à les utiliser.

Nous pouvons avoir des groupes imbriqués (un groupe à l'intérieur d'un autre groupe), et dans ce cas, l'ordre dans lequel les groupes sont fermés doit être cohérent avec l'ordre dans lequel ils ont été ouverts : tout sous-groupe doit être fermé à l'intérieur du groupe dans lequel il a commencé. Il peut également y avoir des groupes vides générés avec la « {} ». Un groupe vide n'a, en principe, aucun effet sur le document final, mais il peut être utile, par exemple, pour indiquer la fin du nom d'une commande.

Le principal effet des groupes est d'encapsuler leur contenu : en règle générale, les définitions, les formats et les attributions de valeurs effectués au sein d'un groupe sont « oubliés » une fois que l'on quitte le groupe. De cette façon, si nous voulons que ConTeXt modifie temporairement son mode de fonctionnement normal, le moyen le plus efficace est de créer un groupe et, au sein de celui-ci, de modifier ce fonctionnement. Ainsi, lorsque nous quitterons le groupe, toutes les valeurs et tous les formats antérieurs à celui-ci seront restaurés. Nous en avons déjà vu quelques exemples en mentionnant des commandes comme `\it`, `\bf`, `\sc`, etc. Mais cela ne se produit pas seulement avec les commandes de formatage : le groupe isole en quelque sorte son contenu, de sorte que toute modification de l'une des nombreuses variables internes que ConTeXt gère en permanence, ne restera effective que tant que nous nous trouvons dans le groupe dans lequel cette modification a eu lieu. De même, une commande définie au sein d'un groupe ne sera pas connue en dehors de celui-ci.

Ainsi, si nous traitons l'exemple suivant

```
\define\A[B]
\A
{
  \define\A[C]
  \A
}
\A
```

B C B

nous voyons que la première fois que nous exécutons la commande `\A`, le résultat correspond à celui de sa définition initiale (« B »). Ensuite, nous avons créé un groupe et redéfini la commande `\A` au sein de celui-ci. Si nous l'exécutons maintenant au sein du groupe, la commande nous donnera la nouvelle définition (« C » dans notre exemple), mais lorsque nous quittons le groupe dans lequel la commande `\A` a été redéfinie, si nous l'exécutons à nouveau, elle tapera « B » une fois de plus. La définition faite au sein du groupe est « oubliée » une fois que nous l'avons quitté.

Une autre utilisation possible des groupes concerne les commandes ou instructions conçues pour s'appliquer exclusivement au caractère qui est écrit après elles. Dans ce cas, si nous voulons que la commande s'applique à plus d'un caractère, nous devons inclure dans un groupe les caractères auxquels nous voulons que la commande ou l'instruction s'applique. Ainsi, par exemple, le caractère réservé « `^` » qui, nous le savons déjà, convertit le caractère suivant en exposant lorsqu'il est utilisé dans l'environnement mathématique ; ainsi, si nous écrivons, par exemple, « `4^2x` », nous obtiendrons « 4^2x ». Mais si nous écrivons « `4^{2x}` », nous obtiendrons « 4^{2x} ».

Enfin, une troisième utilisation du regroupement est d'indiquer à ConTeXt que ce qui est inclus dans le groupe doit être traité comme un seul élément. C'est la raison pour laquelle il a été dit précédemment ([section 3.5](#)) que dans certaines occasions, il est préférable d'enfermer le contenu d'une option de commande entre des crochets.

3.8.2 Dimensions

Bien que nous puissions utiliser ConTeXt parfaitement sans nous soucier des dimensions, nous ne pourrions pas utiliser toutes les possibilités de configuration sans leur accorder une certaine attention. Car, dans une large mesure, la perfection typographique atteinte par TeX et ses dérivés réside dans la grande attention que le système accorde en interne aux dimensions. Les caractères ont des dimensions ; l'espace entre les mots, ou entre les lignes, ou entre les paragraphes ont des dimensions ; les lignes ont des dimensions ; les marges, les en-têtes et les pieds de page. Pour presque tous les éléments de la page auxquels nous pouvons penser, il existe des dimensions.

Dans ConTeXt les dimensions sont indiquées par un nombre décimal suivi par l'unité de mesure. Les unités qui peuvent être utilisées se trouvent dans [table 3.2](#).

Nom	Notation dans ConTeXt	Equivalent
Inch	in	1 in = 2.54 cm
Centimètre	cm	2.54 cm = 1 inch
Millimètre	mm	100 mm = 1 cm
Point	pt	72.27 pt = 1 inch
Big point	bp	72 bp = 1 inch
Scaled point	sp	65536 sp = 1 point
Pica	pc	1 pc = 12 points
Didot	dd	1157 dd = 1238 points
Cicero	cc	1 cc = 12 didots
	ex	
	em	

Tableau 3.2 Unités de mesure dans ConTeXt

Les trois premières unités du [table 3.2](#) sont des mesures standard de longueur ; la première est utilisée dans certaines parties du monde anglophone et les autres en dehors ou dans certaines parties de celui-ci. Les autres unités proviennent du monde de la typographie. Les deux dernières, pour lesquelles je n'ai pas mis d'équivalent, sont des unités de mesure relatives basées sur la police de caractères actuelle. Une « em » est égale à la largeur d'un « M » et une « ex » est égale à la hauteur d'une « x ». L'utilisation de mesures liées à la taille des polices permet de créer des macros qui offrent une mise en forme réussies quelle que soit le contexte d'utilisation puisque tout est mis en cohérence avec la taille de la police de caractère. C'est pourquoi, en général, elle est recommandée.

À de très rares exceptions près, nous pouvons utiliser l'unité de mesure de notre choix, car ConTeXt la convertira en interne. Mais chaque fois qu'une dimension est indiquée, il est obligatoire d'indiquer l'unité de mesure, et même si nous voulons indiquer une mesure de « 0 », nous devons dire « 0pt » ou « 0cm ». Entre le nombre et le nom de l'unité, on peut laisser ou non un espace vide. Si l'unité comporte une partie décimale, nous devons utiliser le « . » en séparateur décimal.

Les mesures sont généralement utilisées comme une option pour une commande. Mais nous pouvons également attribuer directement une valeur à une mesure interne de ConTeXt tant que nous en connaissons le nom. Par exemple :

```
\newdimen\MaDimensionA          % déclaration
\MaDimensionA=10mm              % affectation
\blackrule[width=\MaDimensionA]

\MaDimensionA20mm               % nouvelle affectation
\blackrule[width=\MaDimensionA]

\MaDimensionA 30mm               % nouvelle affectation
\blackrule[width=\MaDimensionA]
```



Nous pouvons utiliser `\MaDimensionA 10mm` (sans le signe égal) mais aussi `\MaDimensionA10mm` sans espace entre le nom de la mesure et sa valeur.

Toutefois, l'attribution d'une valeur directement à une mesure interne est considérée comme une « inelegant ». En général, il est recommandé d'utiliser les commandes qui contrôlent cette variable, et de le faire dans le préambule du fichier source. Le contraire donne lieu à des fichiers sources très difficiles à déboguer car toutes les commandes de configuration ne se trouvent pas au même endroit, et il est vraiment difficile d'obtenir une certaine cohérence dans les caractéristiques typographiques.

Certaines des dimensions utilisées par ConTeXt sont « élastique », c'est-à-dire que, selon le contexte, elles peuvent prendre l'une ou l'autre mesure. Ces mesures sont attribuées avec la syntaxe suivante :

```
\MeasureName Value plus MaxIncrement minus MaxDecrease
```

Par exemple :

```
\parskip 3pt plus 2pt minus 1pt
```

Avec cette instruction, nous demandons à ConTeXt d'attribuer à `\parskip` (indiquant la distance verticale entre les paragraphes qui doit plutôt être paramétrée avec `\setupwhitespace` mais nous avons besoin ici d'un exemple) une mesure *normal* de 3 points, mais que si la composition de la page l'exige, la mesure peut aller jusqu'à 5 points (3 plus 2) ou seulement 2 points (3 moins 1). Dans ces cas, il appartiendra à ConTeXt de choisir la distance pour chaque page entre un minimum de 2 points et un maximum de 5 points.

3.9 Méthode d'auto apprentissage pour ConTeXt

Section ajoutée au dernier moment, lorsque je me suis rendu compte que j'étais moi-même tellement imprégné de l'esprit de ConTeXt que j'étais capable de deviner l'existence de certaines commandes.

L'énorme quantité de commandes et d'options de ConTeXt peut s'avérer vraiment écrasante et nous donner l'impression que nous ne finirons jamais par apprendre à bien travailler avec. Cette impression est trompeuse, car l'un des avantages de ConTeXt est la manière uniforme dont il gère toutes ses structures : en apprenant bien quelques structures, et en sachant, plus ou moins, à quoi servent les autres, lorsque nous aurons besoin d'une fonctionnalité supplémentaire, il sera relativement facile d'apprendre à l'utiliser. C'est pourquoi je pense que cette introduction est une sorte d'*entraînement* qui nous préparera à faire nos propres recherches.

Pour créer un document avec ConTeXt, il suffit probablement de connaître les cinq choses suivantes (nous pourrions les appeler le *Top Five*) :

1. Créer un fichier source ou un projet complet quelconque ; ceci est expliqué dans le [Chapitre 4](#) de cette introduction.
2. Définir la police principale du document, la changer et changer sa couleur ([Chapitre ??](#)).
3. Structurer le contenu de notre document, avec des chapitres, des sections, des sous-sections, etc. Tout cela est expliqué dans le [Chapitre 7](#).
4. Utiliser l'environnement *itemize* pour les listes, il sera expliqué en détail dans [section 12.3](#).
5. ... et à peine plus.

Pour le reste, tout ce dont nous avons besoin, c'est de savoir que c'est possible. Certainement personne n'utilisera une fonctionnalité s'il ne sait pas qu'elle existe. Dans cette introduction, nous expliquons beaucoup d'entre elles ; mais, surtout, il est démontré à plusieurs reprises comment ConTeXt se comporte face à un certain type de construction

- Premièrement, il y aura une commande qui lui permettra de le faire.
- Deuxièmement, il y a presque toujours une commande qui nous permet de configurer et de prédéterminer la façon dont la tâche sera effectuée ; une commande dont le nom commence par `\setup` et coïncide généralement avec la commande de base.
- Enfin, il est souvent possible de créer une nouvelle commande pour effectuer des tâches similaires, mais avec une configuration différente.

Pour savoir si ces commandes existent ou non, consultez la liste officielle des commandes (voir [section 3.6](#)), qui nous informera également des options de configuration que ces commandes prennent en charge. Bien qu'à première vue, les noms de ces options puissent sembler cryptiques, nous verrons rapidement que certaines options sont répétées dans de nombreuses commandes et qu'elles fonctionnent de

la même manière ou de manière très similaire dans toutes ces commandes. Si nous avons des doutes sur ce que fait une option, ou sur son fonctionnement, il suffira de générer un document et de le tester. Nous pouvons également consulter l'abondante documentation de ConTEXt. Comme il est courant dans le monde des logiciels libres, « ConTEXt Standalone » inclut les sources de presque toute sa documentation dans la distribution. Un utilitaire comme « grep » (pour les systèmes GNU Linux) peut nous aider à rechercher si la commande ou l'option sur laquelle nous avons des doutes est utilisée dans l'un de ces fichiers sources afin d'avoir un exemple sous la main.

C'est ainsi que l'apprentissage de ConTEXt a été conçu : l'introduction explique en détail les cinq (en réalité quatre) aspects que j'ai mis en évidence, et bien d'autres encore : au fil de la lecture, une image claire de la séquence se formera dans notre esprit : *une commande pour exécuter la tâche – une seconde commande pour configurer le comportement de la première – une troisième commande pour créer un commande similaire à la première à partir d'un clône*. Nous apprendrons également certaines des principales structures de ConTEXt, et nous saurons à quoi elles servent.

Chapitre 4

Fichiers sources et projets

Table of Contents : 4.1 Codage des fichiers sources ; 4.2 Caractères dans le(s) fichier(s) source(s) que ConTeXt traite d'une manière spéciale ; 4.2.1 Espaces vides (espaces blancs) et tabulations ; 4.2.2 Sauts de ligne ; 4.2.3 Trait d'union et tirets ; 4.3 Projet simple et projet multi-fichiers ; 4.4 Structure du fichier source d'un projet simple ; 4.5 Gestion multi-fichiers à la TeX ; 4.5.1 La commande \input ; 4.5.2 \ReadFile et \readfile ; 4.6 Gestion multi-fichiers à la ConTeXt ; 4.6.1 Fichiers d'environnement ; 4.6.2 Composants et produits ; 4.6.3 Projets ConTeXt ; 4.6.4 Aspects communs des environnements, composants, produits et projets ;

Comme nous le savons déjà, lorsque nous travaillons avec ConTeXt nous commençons toujours par un fichier texte dans lequel sont incluses, outre le contenu du texte, un certain nombre d'instructions indiquant à ConTeXt les transformations qu'il doit appliquer pour générer notre document final correctement formaté en PDF.

En pensant aux lecteurs qui, jusqu'à présent, n'ont su travailler qu'avec des traitements de texte, je pense qu'il vaut la peine de passer un peu de temps avec le fichier source lui-même. Ou plutôt les fichiers sources, car il y a des moments où il n'y a qu'un seul fichier source et d'autres où nous utilisons plusieurs fichiers sources pour arriver au document final. Dans ce dernier cas, nous pouvons parler de « projets multifichiers ».

4.1 Codage des fichiers sources

Le ou les fichiers sources doivent être des fichiers texte. Dans la terminologie informatique, c'est le nom donné à un fichier contenant uniquement du texte lisible par l'homme et ne comportant pas de code binaire. Ces fichiers sont également appelés fichiers *texte simple* ou *texte brut*.

Étant donné qu'en interne, les systèmes informatiques ne traitent que des nombres binaires, un fichier texte est en réalité constitué de *nombres* auxquels sont associés des *caractères*. Une *table de correspondance* est utilisée pour définir cette association entre nombres et caractères. Plusieurs tables peuvent être utilisées. Aussi, le terme *codage d'un fichier texte* fait référence à la table qui est utilisée par le fichier en question.

L'existence de différentes tables de codage pour les fichiers texte est une conséquence de l'histoire de l'informatique elle-même. Aux premiers stades du développement, lorsque la mémoire et la capacité de stockage des dispositifs informatiques étaient rares, il a été décidé d'utiliser une table appelée ASCII (qui signifie « *American Standard Code for Information Interchange* ») (Codage américain standard pour l'échange d'informations) qui n'autorisait que 128 caractères et qui a été établie en 1963 par le comité de normalisation américain. Il est évident que 128 caractères ne suffisent pas à représenter tous les caractères et symboles utilisés dans toutes les langues du monde ; mais c'était plus que suffisant pour représenter l'anglais qui est, de toutes les langues occidentales, celle qui a le moins de caractères, car elle n'utilise pas de diacritiques (accents et autres marques au-dessus ou au-dessous ou à travers d'autres lettres). L'avantage d'utiliser l'ASCII était que les fichiers texte prenaient très peu de place, car 127 (le chiffre le plus élevé de la table) peut être représenté par un nombre binaire à 7 chiffres, et les premiers ordinateurs utilisaient l'octet comme unité de mesure de la mémoire, un nombre binaire à 8 chiffres. Tout caractère de la table peut tenir dans un seul octet. Comme l'octet a 8 chiffres et que l'ASCII n'en utilisait que 7, il restait même de la place pour ajouter d'autres caractères afin de représenter d'autres langues.

Mais lorsque l'utilisation des ordinateurs s'est développée, l'insuffisance de l'ASCII est devenue évidente et il a fallu développer des tables de caractères alternatifs incluant des caractères non connus de l'alphabet anglais, tels que le « ñ » espagnol, les voyelles accentuées, le « ç » catalan ou français, etc. En revanche, il n'y a pas eu d'accord initial sur ce que devaient être ces *tables alternatives* à l'ASCII, de sorte que différentes sociétés informatiques spécialisées se sont progressivement attaquées au problème chacune de leur côté. Ainsi, non seulement des tables spécifiques ont été créées pour différentes langues ou groupes de langues, mais aussi des tables différentes selon la société qui les avait créées (Microsoft, Apple, IBM, etc.).

Ce n'est qu'avec l'augmentation de la mémoire des ordinateurs, la baisse du coût des dispositifs de stockage et l'augmentation correspondante de la capacité que l'idée de créer une table unique pouvant être utilisée pour toutes les langues est apparue. Mais, encore une fois, ce n'est pas une table unique contenant tous les caractères qui a été créée, mais un codage standard (appelé Unicode) ainsi que différentes manières de le représenter (UTF-8, UTF-16, UTF-32, etc.). De tous ces systèmes, celui qui a fini par devenir la norme de facto est l'UTF-8, qui permet de représenter pratiquement toutes les langues vivantes, et de nombreuses langues déjà éteintes, ainsi que de nombreux symboles supplémentaires, le tout en utilisant des nombres de longueur variable (entre 1 et 4 octets), ce qui permet d'optimiser la taille des fichiers texte. Cette taille n'a pas trop augmenté par rapport aux fichiers utilisant l'ASCII pur.

Jusqu'à l'apparition de X_ET_EX les systèmes basés sur T_EX – qui est également né aux États-Unis et a donc l'anglais comme langue maternelle – supposaient que l'codage était en ASCII pur ; ainsi, pour utiliser un codage différent, vous deviez l'indiquer d'une manière ou d'une autre dans le fichier source.

ConT_EXt Mark IV suppose que le codage du fichier source sera UTF-8. Cependant, sur les systèmes informatiques moins récents, un autre codage peut être utilisé par défaut. Je ne suis pas très sûr de l'codage par défaut que par défaut que Windows utilise, étant donné que la stratégie de Microsoft pour atteindre le grand public consiste à cacher la complexité (mais même si elle est cachée, cela ne signifie pas qu'elle a disparu !). Il n'y a pas beaucoup d'informations disponibles (ou je n'ai pas été en mesure de les trouver) concernant le système de codage qu'il utilise par défaut.

Dans tous les cas, quel que soit le codage par défaut, tout éditeur de texte vous permet de d'enregistrer le fichier dans le codage souhaité. Les fichiers sources destinés à être traités par ConT_EXt Mark IV doivent être enregistrés en UTF-8, à moins, bien sûr, il y ait une très bonne raison d'utiliser un autre encodage (bien que je ne puisse pas je ne vois pas quelle pourrait être cette raison).

Si l'on veut écrire un fichier écrit dans un autre codage (peut-être un ancien fichier), nous pouvons :

- Convertir le fichier en UTF-8, option recommandée, et il existe plusieurs outils pour le faire ; sous Linux, par exemple, les commandes `iconv` ou `recode`.
- Indiquer à ConTeXt dans le fichier source que l'encodage n'est pas UTF-8. Pour ce faire, nous devons utiliser la commande `\enablerégime`, dont la syntaxe est `\enablerégime [codage]`, où *codage* fait référence au nom par lequel ConTeXt connaît l'encodage réel du fichier en question. Dans la [table 4.1](#), vous trouverez les différents encodages et les noms par lesquels ConTeXt les connaît.

Codage	Nom pour ConTeXt	Notes
Windows CP 1250	cp1250, windows-1250	Western Europe
Windows CP 1251	cp1251, windows-1251	Cyrillic
Windows CP 1252	cp1252, win, windows-1252	Western Europe
Windows CP 1253	cp1253, windows-1253	Greek
Windows CP 1254	cp1254, windows-1254	Turkish
Windows CP 1257	cp1257, windows-1257	Baltic
ISO-8859-1, ISO Latin 1	iso-8859-1, latin1, il1	Western Europe
ISO-8859-2, ISO Latin 2	iso-8859-2, latin2, il2	Western Europe
ISO-8859-15, ISO Latin 9	iso-8859-15, latin9, il9	Western Europe
ISO-8859-7	iso-8859-7, grk	Greek
Mac Roman	mac	Western Europe
IBM PC DOS	ibm	Western Europe
UTF-8	utf	Unicode
VISCII	vis, viscii	Vietnamese
DOS CP 866	cp866, cp866nav	Cyrillic
KOI8	koi8-r, koi8-u, koi8-ru	Cyrillic
Mac Cyrillic	maccyr, macukr	Cyrillic
Others	cp855, cp866av, cp866mav, cp866tat, ctt, dbk, iso88595, isoir111, milk, mls, mnk, mos, ncc	Various

Tableau 4.1 Références des principales tables de codage pour ConTeXt

ConTeXt Mk IV recommande fortement l'utilisation de UTF-8. Je suis d'accord avec cette recommandation. À partir d'ici dans cette introduction, nous pouvons supposer que l'encodage est toujours UTF-8.

Avec `\enablerégime` ConTeXt inclut la commande `\useregime` qui prend en argument une ou plusieurs références de codage. Je n'ai trouvé aucune information sur cette commande ni sur la façon dont elle diffère de `\enablerégime`, seulement quelques exemples de son utilisation. Je soupçonne que `\useregime` est conçu pour les projets complexes qui utilisent de nombreux fichiers sources, avec l'espoir que tous n'auront pas le même codage. Mais ce n'est qu'une supposition.



4.2 Caractères dans le(s) fichier(s) source(s) que ConTEXt traite d'une manière spéciale

Caractères spéciaux est le nom que je donnerai à un groupe de caractères qui sont différents de *Caractères réservés*. Comme on peut le voir dans [section 3.1](#), ces derniers sont ceux qui ont une signification spéciale pour ConTEXt et ne peuvent donc pas être utilisés directement comme caractères dans le fichier source. En plus de ceux-ci, il existe un autre groupe de caractères qui, bien que traités comme tels par ConTEXt lorsqu'il les trouve dans le fichier source, sont traités avec des règles spéciales. Ce groupe comprend les espaces vides (espaces blancs), les tabulations, les sauts de ligne et les traits d'union.

4.2.1 Espaces vides (espaces blancs) et tabulations

Les tabulations et les espaces blancs sont traités de la même manière dans le fichier source. Un caractère de tabulation (la touche Tab du clavier) sera transformé en espace blanc par ConTEXt. Et les espaces blancs sont absorbés par tout autre espace blanc (ou tabulation) qui les suit immédiatement. Ainsi, cela ne fait absolument aucune différence d'écrire un seul plusieurs espaces et tabulations dans le fichier source :

```
Tweedledum and Tweedledee.
```

ConTEXt considère que ces deux lignes sont exactement les mêmes. Par conséquent, si nous voulons introduire un espace supplémentaire entre les mots, nous devons utiliser certaines commandes ConTEXt qui le font. Normalement, cela fonctionnera avec « \u », c'est-à-dire un caractère \ suivi d'un espace blanc. Mais il existe d'autres procédures qui seront examinées dans le [chapitre 10.3](#) concernant les espacements horizontaux.

```
Dupont et Dupond.
```

```
Dupont\ et\ Dupond.
```

```
Dupont\ \ et\ \ Dupond.
```

```
Dupont\ \ \ et\ \ \ Dupond.
```

```
Dupont et Dupond.
```

L'absorption d'espaces blancs consécutifs nous permet de mettre en forme le fichier source comme nous le souhaitons, par exemple en augmentant ou en diminuant l'indentation utilisée pour le rendre clair et lisible, avec la tranquillité d'esprit de savoir que cela n'affectera en rien le document final. Ainsi, dans l'exemple suivant :

```
The music group from Madrid at the end of the seventies
{\em La Romántica Banda Local}
wrote songs of an eclectic style that were very difficult to categorise.
In their son "El Egipcio", for example, they said:
\quotation{\em
Esto es una farsa más que una comedia,
página muy seria de la histeria musical;
sueños de princesa,
vicios de gitano pueden en su mano acariciar la verdad},
mixing word, phrases simply because they have an internal rhythm
(comedia-histeria-seria, gitano-mano).
```

The music group from Madrid at the end of the seventies *La Romántica Banda Local* wrote songs of an eclectic style that were very difficult to categorise. In their son "El Egipcio", for example, they said: "*Esto es una farsa más que una comedia, página muy seria de la histeria musical; sueños de princesa, vicios de gitano pueden en su mano acariciar la verdad*", mixing word, phrases simply because they have an internal rhythm (comedia-histeria-seria, gitano-mano).

vous pouvez voir que certaines lignes sont légèrement en retrait sur la droite. Ce sont les lignes qui font partie des parties qui apparaîtront en italique. Le fait qu'elles soient en retrait aide (l'auteur) à voir où se termine l'italique.

Certains pourraient penser, quel bazar ! Dois-je m'embêter avec l'indentation des lignes dans mon fichier source ? La vérité est que cette indentation spéciale est faite automatiquement par mon éditeur de texte (GNU Emacs) lorsqu'il édite un fichier source ConTeXt. C'est ce genre de petite aide qui vous fait choisir de travailler avec un certain éditeur de texte et pas un autre.

La règle selon laquelle les espaces vides sont absorbés s'applique exclusivement aux espaces vides consécutifs dans le fichier source. Par conséquent, si un groupe vide (« {} »), est placé dans le fichier source entre deux espaces vides, bien que le groupe vide ne produise rien dans le fichier final, sa présence garantira que les deux espaces vides ne sont pas consécutifs.

La même chose se produit avec le caractère réservé « ~ », bien qu'il ait pour effet de générer un espace blanc alors qu'il n'en est pas vraiment un : un espace blanc suivi d'un ~ ne sera pas absorbé par ce dernier, et un espace blanc après un ~ ne sera pas absorbé non plus.

Ainsi, regardons l'exemple suivant :

Dupont et Dupond.

Dupont {} et Dupond.

Dupont \ et Dupond.

Dupont ~ et Dupond.

Dupont et Dupond.

Dupont et Dupond.

Dupont et Dupond.

Dupont et Dupond.

si vous regardez de près, nous obtenons entre les deux premiers mots, deux espaces consécutifs à la deuxième ligne et trois à la troisième.

4.2.2 Sauts de ligne

Dans la plupart des éditeurs de texte, lorsqu'une ligne dépasse la largeur maximale, un saut de ligne est automatiquement inséré. On peut également insérer expressément un saut de ligne en appuyant sur la touche « Enter » ou « Return ».

ConTeXt applique les règles suivantes aux sauts de ligne :

- a. Un saut de ligne unique est systématiquement équivalent à un espace blanc. Par conséquent, si, immédiatement avant ou après le saut de ligne, il existe un espace blanc ou une tabulation, ceux-ci seront absorbés par le saut de ligne ou le premier espace blanc, et un simple espace blanc sera inséré dans le document final.
- b. Deux ou plusieurs sauts de ligne consécutifs créent un saut de paragraphe. Pour cela, deux sauts de ligne sont considérés comme consécutifs s'il n'y a rien d'autre que des espaces vides ou des tabulations entre le premier et le second saut de ligne (car ceux-ci sont absorbés par le premier saut de ligne) ; ce qui, en résumé, signifie qu'une ou plusieurs lignes consécutives absolument vides dans le fichier source (sans aucun caractère, ou seulement avec des espaces vides ou des tabulations) deviennent un saut de paragraphe.

Notez que j'ai dit « deux ou plusieurs sauts de ligne consécutifs » et ensuite « une ou plusieurs lignes consécutives vides », ce qui signifie que si nous voulons augmenter la séparation entre les paragraphes, nous ne le faisons pas simplement en insérant un autre saut de ligne. Pour cela, nous devrons utiliser une commande qui augmente l'espace vertical. Si nous ne voulons qu'une seule ligne supplémentaire de séparation, nous pouvons utiliser la commande `\blank`. Mais il existe d'autres procédures pour augmenter l'espace vertical. Je vous renvoie à section ??.

Parfois, lorsqu'un saut de ligne devient un espace blanc, nous pouvons nous retrouver avec un espace blanc indésirable et inattendu. En particulier lorsque nous écrivons des macros, où il est facile qu'un espace blanc « s'introduise » sans que nous nous en rendions compte. Pour éviter cela, nous pouvons utiliser le caractère réservé « % » qui, comme nous le savons, fait en sorte que la ligne où il apparaît ne soit pas traitée, ce qui implique que la coupure à la fin de la ligne ne sera pas non plus traitée. Ainsi, par exemple,

```

\define[3]\TestA{%
  {\em #1} {\bf #2} {\sc #3}
}

\define[3]\TestB{%
  {\em #1}
  {\bf #2}
  {\sc #3}
}

\define[3]\TestC{%
  {\em #1}%
  {\bf #2}%
  {\sc #3}%
}

\TestA{riri}{fifi}{loulou}

\TestB{riri}{fifi}{loulou}

\TestC{riri}{fifi}{loulou}

```

riri fifi LOULOU
riri fifi LOULOU
riri**fifi**LOULOU

les commandes `\TestA` et `\TestB` écrivent le premier argument en italique, le second en gras et le troisième en petites capitales, mais la première insérera un espace entre chacun de ces arguments, alors que la seconde n'en n'insérera pas : le caractère réservé `%` empêche les sauts de ligne d'être traités et ils deviennent de simples espaces vides.

4.2.3 Trait d'union et tirets

Les tirets sont un bon exemple de la différence entre un clavier d'ordinateur et un texte imprimé. Sur un clavier normal, il n'existe généralement qu'un seul caractère pour le tiret (ou règle, en termes typographiques) que nous appelons le trait d'union ou (« - ») ; mais un texte imprimé utilise jusqu'à quatre longueurs différentes pour les règles :

- tiret court (ou trait d'union), comme ceux utilisés pour séparer les syllabes dans les césures en fin de ligne (-).
- tiret moyen (ou tiret demi-cadratin), légèrement plus longs que les précédentes (-). Ils ont plusieurs usages dont, pour certaines langues européennes (moins en anglais), lister les énumérations, ou encore séparer les chiffres les moins élevés des chiffres les plus élevés dans une fourchette de dates ou de pages ; [*<pp. 12--33>*].
- tiret longs (ou tiret cadratin) (—), utilisées comme des parenthèses pour inclure une phrase dans une autre.
- signe moins (–) pour représenter une soustraction ou un nombre négatif.

Aujourd'hui, tous les éléments ci-dessus et d'autres encore sont disponibles en encodage UTF-8. Mais comme ils ne peuvent pas tous être générés par une seule touche du clavier, ils ne sont pas si faciles à produire dans un fichier source. Heureusement, TeX a vu la nécessité d'inclure plusieurs traits et tirets dans notre document final que ce qui pouvait être produit par le clavier, et a conçu une procédure simple pour le faire. ConTeXt a complété cette procédure en ajoutant également des

commandes qui génèrent ces différents types de règles. Nous pouvons utiliser deux approches pour générer les quatre types de règles : soit à la manière ordinaire de ConTeXt avec une commande, soit directement à partir du clavier. Ces procédures sont présentées dans [table 4.2](#) :

Type de tiret	Apparence	Écriture directe	Commande
Tiret court / trait d'union	-	-	\hyphen
Tiret moyen / demi-cadratin	--	--	\endash
Tiret long / cadratin	---	---	\emdash
Signe moins	\$-\$	\$-\$	\minus

Tableau 4.2 Rules/dashes in ConTeXt

Les noms de commandes \hyphen et \minus sont ceux normalement utilisés en anglais. Bien que de nombreux professionnels de l'imprimerie les appellent « tirets », les termes de TeX, à savoir \endash et \emdash sont également courants dans la terminologie de la composition. Les « *en* » et « *em* » sont les noms des unités de mesure utilisées en typographie. Une « *en* » représente la largeur d'un « *n* » tandis que « *em* » est la largeur d'un « *m* » dans la police utilisée.

4.3 Projet simple et projet multi-fichiers

En ConTeXt nous pouvons utiliser un seul fichier source qui inclut absolument tout le contenu de notre document final ainsi que tous les détails s'y rapportant, dans ce cas nous parlons de « projet simple », ou, au contraire, nous pouvons utiliser un certain nombre de fichiers sources qui partagent le contenu de notre document final, et dans ce cas nous parlons de « projet multi-fichier ».

Les scénarios dans lesquels il est typique de travailler avec plus d'un fichier source sont les suivants :

- Si nous écrivons un document dans lequel plusieurs auteurs ont collaboré, chacun ayant sa propre partie différente des autres ; par exemple, si nous écrivons un livre hommage avec des contributions de différents auteurs, ou un numéro de magazine, etc.
- Si nous sommes en train d'écrire un long document où chaque partie (chapitre) a une autonomie relative, de sorte que l'arrangement final de ceux-ci permet plusieurs possibilités et sera décidé à la fin. Cela se produit assez fréquemment pour de nombreux textes académiques (manuels, introductions, etc.) où l'ordre des chapitres peut varier.
- Si nous rédigeons un certain nombre de documents connexes qui partagent certaines caractéristiques de style ou macro, il est alors intéressant de rassembler ces éléments dans des fichiers séparés, et ainsi utilisables directement dans d'autres projets. Ces fichiers constituent comme des modèles de composition de document.
- Si, tout simplement, le document sur lequel nous travaillons est volumineux, de sorte que l'ordinateur ralentit soit lors de l'édition, soit lors de la compilation ; dans ce cas, le fait de répartir le matériel sur plusieurs fichiers sources accélérera considérablement la compilation de chacun.

4.4 Structure du fichier source d'un projet simple

In simple projects developed in a single source file, the structure is very simple and revolves around the « text » environment that must essentially appear in the same file. We differentiate between the following parts of this file :

- **Le préambule du document** : tout ce qui va de la première ligne du fichier jusqu'au début de l'environnement « text » indiqué par la commande (`\starttext`).
- **Le corps du document** : il s'agit du contenu de l'environnement « text » ; ou en d'autres termes, tout ce qui se trouve entre `\starttext` et `\stoptext`.

```
% Première ligne du document

% Zone Préambule:
% Contenant la configuration globales des commandes
% pour l'ensemble du document

\starttext    % Début du contenu à proprement parler

...
...          % Contenu du document
...

\stoptext    % Fin du contenu le reste sera ignoré
```

Figure 4.1 Fichier source contenant un projet simple

Dans figure 4.1 nous voyons un fichier source très simple. Absolument tout ce qui précède la commande `\starttext` (qui, dans l'image, se trouve à la ligne 5, en ne comptant que celles contenant du texte), constitue le préambule ; tout ce qui se trouve entre `\starttext` et `\stoptext` constitue le corps du document. Tout ce qui suit le stoptext sera ignoré.

Le préambule est utilisé pour inclure les commandes qui affectent le document dans son ensemble, celles qui déterminent sa configuration générale. Il n'est pas indispensable d'écrire une commande dans le préambule. S'il n'y en a pas, ConTeXt adoptera une configuration par défaut qui n'est pas très développée mais qui pourrait faire l'affaire pour de nombreux documents. Dans un document bien planifié, le préambule contiendra toutes les commandes affectant le document dans son ensemble, comme les macros et les commandes personnalisées à utiliser dans le fichier source. Dans un préambule typique, cela pourrait inclure les éléments suivants :

- la langue principale du document (Voir section 10.5).
- le format du papier (section ??) et de la mise en page (section 5.3).
- la police principale des documents (section 6.3).
- la personnalisation des commandes de section à utiliser (section 7.4) et, le cas échéant, la définition de nouvelles commandes de section (section 7.5).
- les en-têtes et les pieds de page (section 5.6).
- les paramètres pour nos propres macros (section 3.7).
- Etc.

Le préambule est destiné à la configuration générale du document ; par conséquent, rien de ce qui concerne le contenu du document ou le texte à traiter ne doit y figurer. En théorie, tout texte traitable inclus dans le préambule sera ignoré, bien que parfois, s'il est présent, il provoquera une erreur de compilation.

Le corps du document, encadré entre les commandes `\starttext` et `\stoptext` comprend le contenu réel, c'est-à-dire le texte réel, ainsi que les commandes ConTeXt qui s'applique à des parties spécifiques du texte contenu et non à l'ensemble du document.

4.5 Gestion multi-fichiers à la *TEX*

Afin de pouvoir travailler avec plus d'un fichier source, *TEX* a inclus la primitive appelée `\input`, qui fonctionne également dans ConTeXt, bien que ce dernier comprenne deux commandes spécifiques qui comme nous allons le voir, perfectionnent dans une certaine mesure le fonctionnement de `\input`.

4.5.1 La commande `\input`

La commande `\input` insère le contenu du fichier qu'elle indique. Son format est le suivant :

```
\input NomFichier
```

où *NomFichier* est le nom du fichier à insérer. Notez qu'il n'est pas nécessaire de placer le nom du fichier entre des accolades, bien que la commande ne génère pas d'erreur si cela est fait. En revanche, il ne doit jamais être placé entre crochets. Si l'extension du fichier est « `.tex` », elle peut être omise.

Lorsque ConTeXt compile un document et trouve une commande `\input`, il recherche le fichier indiqué et poursuit la compilation comme si ce fichier faisait partie du fichier qui l'a appelé. Lorsqu'il a terminé la compilation, il retourne au fichier d'origine et reprend là où il s'est arrêté ; le résultat pratique est donc que le contenu du fichier appelé au moyen de `\input` est inséré à l'endroit où il est appelé. Le fichier appelé par `\input` doit avoir un nom valide dans notre système d'exploitation et ne doit pas comporter d'espaces vides. ConTeXt le cherchera dans le répertoire de travail, et s'il ne le trouve pas là, il le cherchera dans les répertoires inclus dans la variable de l'environnement `TEXROOT`. Si le fichier n'est finalement pas trouvé, il produira une erreur de compilation.

L'utilisation la plus courante de la commande `\input` est la suivante : un fichier est écrit, appelons-le « `principal.tex` », et il sera utilisé comme conteneur pour appeler, par le biais de la commande `\input`, les différents fichiers qui composent notre projet. Ceci est illustré dans l'exemple suivant :

```
\input MaConfiguration % Commandes de configuration générale

\starttext

\input PageDeTitre
\input Preface

\input Chap1
\input Chap2
...
\input Chap6

\stoptext
```

Notez comment, pour la configuration générale du document, nous avons appelé le fichier « MaConfiguration.tex » qui, nous le supposons, contient les commandes globales que nous voulons appliquer. Ensuite, entre les commandes `\starttext` et `\stoptext` nous appelons les différents fichiers qui contiennent le contenu des différentes parties de notre document. Si, à un moment donné, pour accélérer le processus de compilation, nous souhaitons ne pas compiler certains fichiers, il suffit de mettre une marque de commentaire au début de la ligne appelant ce ou ces fichiers. Par exemple, si nous sommes en train d'écrire le deuxième chapitre et que nous voulons le compiler simplement pour vérifier qu'il ne contient pas d'erreurs, nous n'avons pas besoin de compiler le reste et pouvons donc écrire :

```
\input MaConfiguration % Commandes de configuration générale  
\starttext  
% \input PageDeTitre  
% \input Preface  
  
% \input Chap1  
  \input Chap2  
  ...  
% \input Chap6  
  
\stoptext
```

et seul le chapitre 2 sera compilé. Notez comment, d'autre part, changer l'ordre des chapitres est aussi simple que de changer l'ordre des lignes qui les appellent.

Lorsque nous excluons un fichier de la compilation d'un projet multi-fichier, nous gagnons en vitesse de traitement, mais parmi les conséquences, toutes les références que la partie en cours de compilation fait à d'autres parties non encore compilées ne fonctionneront plus. Voir [section 9.2](#).

Il est important de préciser que lorsque nous travaillons avec `\input`, seul le fichier principal, celui qui appelle tous les autres, doit inclure les commandes `\starttext` et `\stoptext`, car si les autres fichiers les incluent, il y aura une erreur. Cela signifie, d'autre part, que nous ne pouvons pas compiler directement les différents fichiers qui composent le projet, mais que nous devons nécessairement les compiler à partir du fichier principal, qui est celui qui abrite la structure de base du document.

4.5.2 `\ReadFile` et `\readfile`

Comme nous venons de le voir, si ConTeXt ne trouve pas le fichier appelé avec `\input`, il génère une erreur. Dans le cas où nous voulons insérer un fichier uniquement s'il existe, mais en tenant compte de la possibilité qu'il n'existe pas, ConTeXt offre une variante de la commande `\input` :

```
\ReadFile{MonFichier}
```

Cette commande est en tous points similaire à `\input`, à la seule exception que si le fichier à insérer est introuvable, elle poursuivra la compilation sans générer d'erreur. Elle diffère également de `\input` par sa syntaxe, puisque nous savons qu'avec `\input` il n'est pas nécessaire de mettre le nom du fichier à insérer entre accolades. Mais avec `\ReadFile`, c'est nécessaire.

Si nous n'utilisons pas d'accolades, ConTeXt pensera que le nom du fichier à rechercher est le même que le premier caractère qui suit la commande `\ReadFile`, suivi de l'extension `.tex`. Ainsi, par exemple, si nous écrivons

```
\ReadFile MonFichier
```

ConTeXt comprendra que le fichier à lire s'appelle « `M.tex` », puisque le caractère qui suit immédiatement la commande `\ReadFile` (à l'exception des espaces vides qui sont, comme nous le savons, ignorés à la fin du nom d'une commande) est une « `M` ». Étant donné que ConTeXt ne trouvera normalement pas un fichier appelé « `M.tex` », et que `\ReadFile` ne génère pas d'erreur s'il ne trouve pas le fichier, ConTeXt continuera la compilation après le « `M` » dans « `MonFichier` », et insérera le texte « `onFichier` ».

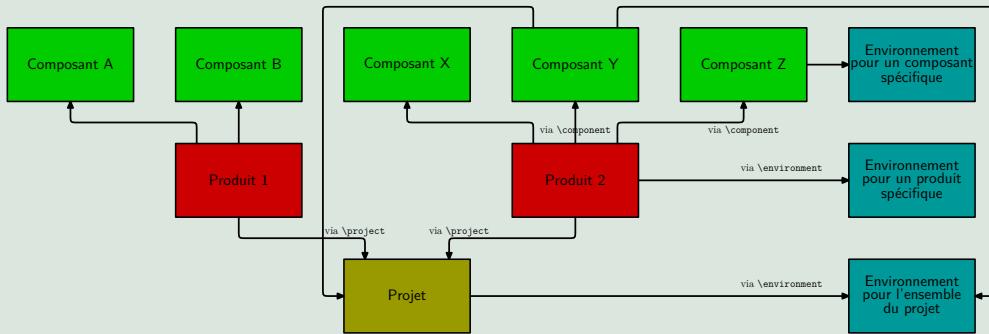
Une version plus raffinée de `\ReadFile` est `\readfile` dont le format est le suivant :

```
\readfile{MonFichier}{àInsérerSiExistant}{àInsérerSiNonExistant}
```

Le premier argument est similaire à `\Readfile` : le nom d'un fichier entre accolades. Le deuxième argument comprend le texte à écrire si le fichier existe, avant d'insérer le contenu du fichier. Le troisième argument comprend le texte à écrire si le fichier en question n'est pas trouvé. Cela signifie que, selon que le fichier saisi comme premier argument est trouvé ou non, le deuxième argument (si le fichier existe) ou le troisième (si le fichier n'existe pas) sera exécuté.

4.6 Gestion multi-fichiers à la ConTEXt

Le troisième mécanisme que propose ConTEXt pour les projets multi-fichiers est plus complexe et plus complet : il commence par faire une distinction entre différents fichiers : les fichiers de projet, les fichiers de produit, les fichiers de composants et les fichiers d'environnement.



Pour comprendre les relations et le fonctionnement de chacun de ces types de fichiers, je pense qu'il est préférable de les expliquer individuellement :

4.6.1 Fichiers d'environnement

Un fichier d'environnement est un fichier qui stocke les macros et les configurations d'un style spécifique destinées à être appliquées à plusieurs documents, qu'il s'agisse de documents totalement indépendants ou de parties d'un document complexe. Le fichier d'environnement peut donc inclure tout ce que nous écririons normalement avant `\starttext`, c'est-à-dire la configuration générale du document.

J'ai conservé le terme « fichiers d'environnement » pour ces types de fichiers, afin de ne pas m'écartez de la terminologie officielle de ConTEXt même si je pense qu'un meilleur terme serait probablement « fichiers de mise en forme » ou « fichiers de configuration générale ».

Comme tous les fichiers source de ConTEXt, les fichiers d'environnement sont des fichiers texte, et supposent que l'extension sera « `.tex` », bien que si nous le voulons, nous pouvons la changer, peut-être en « `.env` ». Cependant, cela n'est généralement pas fait dans ConTEXt. Le plus souvent, les fichiers d'environnement sont identifiés en commençant ou en terminant leur nom par « `env` ». Par exemple : « `env_livreA.tex` » ou « `mon-livreA.tex` ». L'intérieur d'un tel fichier d'environnement ressemblerait à ce qui suit :

```
\startenvironment env_livreA

\mainlanguage[fr]

\setupbodyfont
  [palatino,14pt]

\setupwhitespace
  [big]

...

\stopenvironment
```

ou pourrait également faire appel à d'autres fichiers environnements de façon à décomposer les différents éléments :

```
\startenvironment env_livreA

\environment  env_livreA-polices
\environment  env_livreA-couleurs
\environment  env_livreA-abbreviations
\environment  env_livreA-urls
\environment  env_livreA-macros

\stopenvironment
```

En d'autres termes, les définitions et les commandes de configuration se trouvent dans `\startenvironment` et `\stopenvironment`. Immédiatement après `\startenvironment`, nous écrivons le nom par lequel nous voulons identifier l'environnement en question, puis nous incluons toutes les commandes dont nous aimerais que notre environnement soit composé.

En ce qui concerne le nom de l'environnement, d'après mes tests, le nom que nous ajoutons immédiatement après `\startenvironment` est simplement indicatif, et si nous ne lui donnions pas de nom, alors rien de préjudiciable ne se passe.

Les fichiers d'environnement ont été conçus pour fonctionner avec des composants et des produits (expliqués dans la section suivante). C'est pourquoi un ou plusieurs environnements peuvent être appelés depuis un composant ou un produit à l'aide de la commande `\environment`. Mais cette commande fonctionne également si elle est utilisée dans la zone de configuration (préambule) de tout fichier source ConTeXt, même s'il ne s'agit pas d'un fichier source destiné à être compilé en parties.

La commande `\environment` peut être appelée en utilisant l'un des deux formats suivants :

```
\environment env_livreA
\environment [env_livreA]
```

Dans les deux cas, l'effet de cette commande sera de charger le contenu du fichier pris en argument. Si ce fichier n'est pas trouvé, la compilation se poursuivra de manière normale sans générer d'erreur. Si l'extension du fichier est « `.tex` », elle peut être omise.

4.6.2 Composants et produits

Dans le cas d'un livre dont chaque chapitre se trouve dans un fichier source différent, on dira que les chapitres sont des *composants* et que le livre est le *produit*. Cela signifie que le composant est une partie autonome d'un produit, capable d'avoir son propre style et d'être compilé indépendamment. Chaque composant aura un fichier différent et, en outre, il y aura un fichier produit qui rassemblera tous les composants. Les commandes utilisées `\startcomponent` et `\startproduct` sont suivies d'un nom qui sert à se repérer mais qui n'a pas d'impact sur le fonctionnement de ConTEXt. L'habitude consiste à indiquer le nom du fichier lui-même.

Un fichier de composant typique « `cmp_chapitre-1.tex` » serait le suivant

```
\startcomponent cmp_chapitre-1
  \startchapter[title={Titre du chapitre 1}]
  ...
\stopcomponent
```

Et un fichier produit « `prd_livre-A.tex` » rassemblant les composants ressemblerait à ce qui suit :

```
\startproduct prd_livre-A
  \environment env_livreA
  \component cpm_chapitre-1
  \component cpm_chapitre-2
  \component cpm_chapitre-3
  ...
\stopproduct
```

Le nom du composant qui est appelé à partir d'un produit doit être le nom du fichier qui contient le composant en question. Toutefois, si l'extension de ce fichier est « `.tex` », il peut être omis.

Pour les questions concernant les chemins de fichiers et répertoire voyez le paragraphe dédié [page 101](#).

Notez que le contenu réel de notre document sera réparti entre les différents fichiers « component » et que le fichier produit se limite à établir l'ordre des composants. D'autre part, les composants (individuels) et les produits peuvent être compilés directement. La compilation d'un produit génère un fichier PDF contenant tous les composants de ce produit. Si, par contre, l'un des composants est compilé individuellement, cela générera un fichier PDF contenant uniquement le composant compilé.

Dans un fichier de composant, nous pouvons appeler un ou plusieurs fichiers d'environnement avec `\environment FichierEnvironment`. Nous pouvons faire de même dans le fichier produit. Plusieurs fichiers d'environnement peuvent être chargés simultanément. Nous pouvons, par exemple, avoir notre collection préférée de macros et les différents styles que nous appliquons à nos documents dans différents fichiers. Notez toutefois que lorsque nous utilisons deux ou plusieurs environnements, ceux-ci sont chargés dans l'ordre dans lequel ils sont appelés, de sorte que si la même commande de configuration a été incluse dans plusieurs environnements et qu'elle a des valeurs différentes, ce sont les valeurs du dernier environnement chargé qui s'appliquent. D'autre part, les fichiers d'environnement ne sont chargés qu'une seule fois, donc dans les exemples précédents où l'environnement est appelé à partir du fichier produit et de fichiers composants spécifiques, si nous compilons le produit, c'est le moment où les environnements sont chargés, et dans l'ordre indiqué ; quand un environnement est appelé à partir de l'un des composants, ConTeXt vérifiera si cet environnement est déjà chargé, auquel cas il ne fera rien.

Si le fichier composant ne fait référence à aucun fichier environnement (ou fichier projet comme nous le verrons juste après), sa compilation directe n'appliquera pas les environnements appelés par le fichier produit.

Au contraire, si le fichier composant fait référence à un fichier environnement spécifique, la compilation du produit les appliquera.

4.6.3 Projets ConTeXt

La distinction entre produits et composants est suffisante dans la plupart des cas. Néanmoins, ConTeXt possède un niveau encore plus élevé où l'on peut regrouper un certain nombre de produits : il s'agit du *projet*.

Un fichier projet typique se présenterait plus ou moins comme suit

```
\startproject  prj_macollection
\environment   env_general
\product      prd_livre-A % version française
\product      prd_livre-B % version espagnole
\product      prd_livre-C % version anglaise
...
\stopproject
```

Un scénario dans lequel nous aurions besoin d'un projet serait, par exemple, lorsque nous devons éditer une collection de livres, tous avec les mêmes spécifications de format ; ou bien différentes traduction d'un même livre ; ou si nous éditons une revue : la collection de livres, ou la revue en tant que telle, serait le projet ; chaque livre ou chaque numéro de revue serait un produit ; et chaque chapitre d'un livre ou chaque article d'un numéro de revue serait un composant.

Les projets, en revanche, ne sont pas destinés à être compilés directement. Considérons que, par définition, chaque produit appartenant au projet (chaque livre de la collection, ou chaque numéro de revue) doit être compilé séparément et générer son propre PDF. Par conséquent, la commande `\product` incluse dans le projet pour indiquer quels produits appartiennent au projet, ne fait en fait rien : il s'agit simplement d'un rappel pour l'auteur.

Évidemment, certains pourraient demander pourquoi nous avons des projets s'ils ne peuvent pas être compilés : la réponse est que le fichier de projet lie certains environnements au projet. C'est pourquoi, si nous incluons la commande `\projet NomProjet` dans un fichier de composant ou de produit, ConTEXt lira le fichier de projet et chargera automatiquement les environnements qui lui sont liés. C'est pourquoi la commande `\environnement` dans les projets doit venir après `\start-project` (comme pour les composants et produits)

Tout comme pour les commandes `\environnement` et `\composant`, la commande `\projet` nous permet d'indiquer le nom du projet entre crochets ou de ne pas utiliser de crochets du tout. Cela signifie que `\projet NomdeFichier` et `\Projet[NomdeFichier]` sont des commandes équivalentes. **Résumé des différentes manières de charger un environnement** Il ressort de ce qui précède qu'un environnement peut être chargé par n'importe laquelle des procédures suivantes :

- a. Pour un fichier composant, en insérant entre `\startcomponent` et `\start-text` soit la commande `\environnement FichierEnvironnement` soit la commande `\projet FichierProjet`.
- b. Pour un fichier produit, en insérant entre `\startproduct` et le premier `\component` soit la commande `\environnement FichierEnvironnement` soit la commande `\projet FichierProjet`.

4.6.4 Aspects communs des environnements, composants, produits et projets

Noms des environnements, des composants, des produits et des projets : Nous avons déjà vu que, pour tous ces éléments, après la commande `\start` qui initie un environnement, un composant, un produit ou un projet particulier, son nom est saisi directement. Ce nom, en règle générale, doit coïncider avec le nom du fichier contenant l'environnement, le composant ou le produit car, par exemple, lorsque ConTEXt est en train de compiler un produit et que, selon le fichier du produit, il doit charger un environnement ou un composant, nous n'avons aucun moyen de savoir quel est le fichier de cet environnement ou de ce composant, à moins que le fichier ait le même nom que l'élément à charger.

Dans le cas contraire, d'après mes tests, le nom écrit après `\startproduct`, `\startcomponent`, `\startproject` ou `\startenvironment` dans les fichiers produit et environnement est simplement indicatif. S'il est omis une erreur bloque la compilation, mais s'il ne correspond pas au nom du fichier, rien de grave ne se produit.

Structure des répertoires et chemins des fichiers :

Par défaut, ConTeXt cherche les fichiers dans le répertoire de travail et dans le chemin indiqué par la variable TEXROOT. Cependant, lorsque nous utilisons les commandes `\project`, `\product`, `\component` ou `\environment`, il est supposé que le projet possède une structure de répertoires dans laquelle les éléments communs se trouvent dans le répertoire parent, et les éléments spécifiques dans un répertoire enfant. Ainsi, si le fichier indiqué dans le répertoire de travail est introuvable, il sera recherché dans son répertoire parent, et s'il n'est pas trouvé là non plus, dans le répertoire parent de ce répertoire, et ainsi de suite.

Il est parfois utile d'indiquer le chemin d'un fichier particulier, cela se fait naturellement par exemple :

```
\component chapitres/cmp_chapitre-6
```

Mais bien souvent cela se définit directement dans l'un des fichiers environnements par la commande `\usepath` qui permet d'indiquer la liste des répertoires dans lesquels ConTeXt doit chercher les fichiers .tex.

```
\usepath [chapitres, annexes]
```

Une autre commande `\setupexternalfigures` permet d'indiquer le chemin des images (dont l'utilisation sera détaillée à la section 13.2), avec une syntaxe similaire indiquée à l'option `directory` :

```
\setupexternalfigures [directory={../general_images, images}]}
```

II

Composition des éléments généraux au document

Chapitre 5

Pages et pagination

Table of Contents : 5.1 Taille de la page ; 5.1.1 Réglage du format de la page ; 5.1.2 Utiliser des dimensions non-standard ; A Syntaxe alternative de `\setpapersize` ; B Définition d'un format de page personnalisé ; 5.1.3 Modification du format de la page à n'importe quel endroit du document ; 5.1.4 Adapter la taille de la page à son contenu ; 5.2 Éléments sur la page ; 5.3 Mise en page (`\setuplayout`) ; 5.3.1 Attribution d'une dimension aux différents composants de la page ; 5.3.2 Adapter la mise en page ; 5.3.3 Utilisation de différentes mises en page ; 5.3.4 Autres questions relatives à la mise en page ; A Distinction entre les pages paires et impaires ; B Pages avec plus d'une colonne ; 5.4 Numérotation des pages ; 5.5 Sauts de page forcés ou suggérés ; 5.5.1 La commande `\page` ; 5.5.2 Joindre certaines lignes ou certains paragraphes pour empêcher l'insertion d'un saut de page entre eux ; 5.6 En-têtes et pieds de page ; 5.6.1 Commandes pour établir le contenu des en-têtes et des pieds de page ; 5.6.2 Mise en forme des en-têtes et des pieds de page ; 5.6.3 Définir des en-têtes et des pieds de page spécifiques et les relier aux commandes de section ; 5.7 Insertion d'éléments de texte dans les bords de page et les marges ;

ConTeXt transforme le document source en *pages* correctement formatées. L'aspect de ces pages, la façon dont le texte et les espaces vides sont répartis et les éléments qu'elles comportent sont autant d'éléments fondamentaux pour une bonne composition. Ce chapitre est consacré à toutes ces questions, ainsi qu'à d'autres sujets relatifs à la pagination.

5.1 Taille de la page

5.1.1 Réglage du format de la page

Par défaut, ConTeXt suppose que les documents seront de format A4, la norme européenne. On peut établir un format différent avec `\setpapersize` qui est la commande typique que l'on trouve dans le préambule du fichier source. La syntaxe *normal* de cette commande est :

```
\setpapersize[FormatPageLogique,Orientation][FormatPagePhysique,Orientation]
```

où les deux arguments prennent des noms symboliques.¹³ Le premier argument, que j'ai appelé *FormatPageLogique*, représente la taille de la page finale qui est celle à prendre en compte pour sa composition ; et le second argument, *FormatPagePhysique*, représente la taille de la page de papier sur laquelle les pages du document

¹³ Rappelez-vous que dans la section 3.5 j'ai indiqué que les options prises par les commandes ConTeXt sont essentiellement de deux sortes : des noms symboliques, dont la signification est déjà connue de ConTeXt, ou des variables auxquelles nous devons explicitement attribuer une valeur.

seront imprimées puis découpées. Bien souvent (document purement numérique, impression à la maison ou au bureau) les deux tailles sont identiques et le second argument peut être omis. Cependant, il arrive que les deux tailles soient différentes, comme par exemple lors de l'impression d'un livre en feuilles de 8 ou 16 pages (une technique d'impression courante, notamment pour les livres universitaires jusqu'aux années 1960 environ). Dans ces cas, ConTEXt nous permet de distinguer les deux tailles ; et si l'idée est d'imprimer plusieurs pages sur une seule feuille de papier, nous pouvons également indiquer le schéma de pliage à suivre en utilisant la commande `\setuparranging` (qui ne sera pas expliquée dans cette introduction).

Pour le format de composition, nous pouvons indiquer n'importe lequel des formats prédéfinis utilisés par l'industrie du papier (ou par nous-mêmes). Cela inclut :

- Toute série A, B et C définie par la norme ISO-216 (de A0 à A10, de B0 à B10 et de C0 à C10), généralement utilisée en Europe.
- S3 - S6, S8, SM, SW pour les tailles d'écran dans les documents non destinés à être imprimés mais affichés à l'écran.
- N'importe quel format standard américain : lettre, ledger, tabloid, légal, folio, executive.
- L'un des formats RA et RSA définis par la norme ISO-217.
- Les formats G5 et E5 définis par la norme suisse SIS-014711 (pour les thèses de doctorat).
- Pour les enveloppes : l'un des formats définis par la norme nord-américaine (enveloppe 9 à enveloppe 14) ou par la norme ISO-269 (C6/C5, DL, E4).
- CD, pour les pochettes de CD.

En même temps que le format du papier, avec `\setuppapersize` on peut indiquer l'orientation de la page : « portrait » (verticale) ou « landscape »(horizontale).

Les autres options que `\setuppapersize` permet, selon le wiki ConTEXt, sont « `rotated` », « `90` », « `180` », « `270` », « `mirrored` » et « `negative` ». Dans mes propres tests, je n'ai remarqué que quelques changements perceptibles avec « `rotated` » qui inverse la page, bien que ce ne soit pas exactement une inversion. Les valeurs numériques sont censées produire le degré de rotation équivalent, seules ou en combinaison avec « `rotated` », mais je n'ai pas réussi à les faire fonctionner. Je n'ai pas non plus découvert exactement à quoi servent « `mirrored` » et « `negative` ».

Le deuxième argument de `\setuppapersize`, dont j'ai déjà dit qu'il peut être omis lorsque la taille d'impression n'est pas différente de la taille de composition, peut prendre les mêmes valeurs que le premier, indiquant la taille et l'orientation du papier. Elle peut également prendre la valeur « `oversized` » qui, selon le wiki ConTEXt ajoute un centimètre et demi à chaque coin du papier.

Selon le wiki, il existe d'autres valeurs possibles pour le deuxième argument : « `sous-format` », « `double-format` » et « `double-surformat` ». Mais lors de mes propres tests, je n'ai constaté aucun changement après l'utilisation de l'une de ces valeurs ; la définition officielle de la commande (voir section 3.6) ne mentionne pas non plus ces options.

5.1.2 Utiliser des dimensions non-standard

Si nous voulons utiliser une taille de page non standard, il y a deux choses que nous pouvons faire :

1. Utiliser une syntaxe alternative de `\setuppapersize` qui nous permet d'introduire la hauteur et la largeur du papier comme dimensions.
2. Définir notre propre format de page, en lui attribuant un nom et en l'utilisant comme s'il s'agissait d'un format de papier standard.

A. Syntaxe alternative de `\setuppapersize`

Outre la syntaxe que nous avons déjà vue, `\setuppapersize` nous permet d'utiliser cette autre syntaxe :

```
\setuppapersize[Nom][Options]
```

où *Nom* est un argument facultatif qui représente le nom attribué à un format de papier avec `\definepapersize` (que nous verrons ensuite), et *Options* est l'attribution d'une valeur explicite. Parmi les options autorisées, nous pouvons souligner les suivantes :

- `width`, `height` qui représentent, respectivement, la largeur et la hauteur de la page.
- `page`, `paper`. Le premier fait référence à la taille de la page à composer, et le second à la taille de la page à imprimer physiquement. Cela signifie que « `page` » est équivalent au premier argument de `\setuppapersize` dans sa syntaxe normale (expliquée ci-dessus) et « `paper` » au second argument. Ces options peuvent prendre les mêmes valeurs que celles indiquées précédemment (A4, S3, etc.).
- `scale`, indique un facteur d'échelle pour la page.
- `topspace`, `backspace`, `offset`, distances supplémentaires.

B. Définition d'un format de page personnalisé

Pour définir un format de page personnalisé, on utilise la commande `\definepapersize`, dont la syntaxe est la suivante

```
\definepapersize[Name][Options]
```

où *Nom* désigne le nom donné au nouveau format et *Options* peut être :

- Toute valeur autorisée pour `\setuppapersize` dans sa syntaxe normale (A4, A3, B5, CD, etc.).
- Mesures de la largeur (du papier), de la hauteur (du papier) et du décalage (déplacement), ou une valeur mise à l'échelle (« `scale` »).

Il n'est pas possible de mélanger les valeurs autorisées pour `\setuppapersize` avec des mesures ou des facteurs d'échelle. En effet, dans le premier cas, les options sont des mots symboliques et dans le second, des variables auxquelles on donne une valeur explicite ; et dans ConTeXt, comme je l'ai déjà dit, il n'est pas possible de mélanger les deux types d'options.

Lorsque nous utilisons `\definepapersize` pour indiquer un format de papier qui coïncide avec certaines des mesures standard, en fait, plutôt que de définir un nouveau format de papier, ce que nous faisons est de définir un nouveau nom pour un format de papier déjà existant. Cela peut être utile si nous voulons combiner un format de papier avec une orientation. Ainsi, par exemple, nous pouvons écrire

```
\definepapersize[vertical][A4, portrait]
\definepapersize[horizontal][A4, landscape]
```

5.1.3 Modification du format de la page à n'importe quel endroit du document

Dans la plupart des cas, les documents n'ont qu'un seul format de page et c'est pourquoi `\setuppapersize` est la commande typique que nous incluons dans le préambule et que nous n'utilisons qu'une seule fois dans chaque document. Cependant, à certaines occasions, il peut être nécessaire de changer la taille à un moment donné du document ; par exemple, si à partir d'un certain point une annexe est incluse dans laquelle les feuilles sont en paysage.

Dans ce cas, nous pouvons utiliser `\setuppapersize` à l'endroit précis où nous voulons que le changement se produise. Mais comme le format change immédiatement, pour éviter des résultats inattendus, il faut normalement insérer un saut de page forcé avant le `\setuppapersize`.

Si nous n'avons besoin de modifier la taille de la page que pour une seule page, au lieu d'utiliser deux fois `\setuppapersize`, une fois pour passer à la nouvelle taille et la seconde pour revenir à la taille d'origine, nous pouvons utiliser `\adaptpapersize` qui modifie la taille de la page et, une page plus tard, réinitialise automatiquement la valeur avant d'être appelée. De la même manière qu'avec `\setuppapersize`, nous devons insérer un saut de page forcé avant d'utiliser `\adaptpapersize`.

5.1.4 Adapter la taille de la page à son contenu

Il existe trois environnements dans ConTeXt qui génèrent des pages strictement limitée à la taille exacte de leur contenu. Il s'agit de `\startMPpage`, `\startpagefigure` et `\startTEXpage`. La première génère une page qui contient un graphique généré avec MetaPost, un langage de conception graphique qui s'intègre à ConTeXt, mais dont je ne parlerai pas dans cette introduction. La seconde fait la même chose avec une image et peut-être un peu de texte en dessous. Elle prend deux arguments : le premier identifie le fichier contenant l'image. J'en parlerai dans le chapitre consacré aux images externes page ???. La troisième (`\startTEXpage`) contient le texte qui est son contenu sur une page. Sa syntaxe est la suivante :

```
\startTEXpage[Options] ... \stopTEXpage
```

où les options peuvent être l'une des suivantes :



- **strut**. Je ne suis pas sûr de l'utilité de cette option. Dans la terminologie de ConTeXt un *strut* est un caractère sans largeur, mais avec une hauteur maximale et une profondeur , mais je ne vois pas bien ce que cela a à voir avec l'utilité globale de cette commande. Selon le wiki, cette option permet les valeurs « yes », « no », « global » et « local », et où la valeur par défaut est « no ».
- **align**. Indique l'alignement du texte. Il peut s'agir de « normal », « flush-left », « flushright », « middle », « high », « low » ou « lohi ».
- **offset** pour indiquer la quantité d'espace blanc autour du texte. Il peut s'agir de « none », « overlay » si un effet de superposition est souhaité, ou d'une dimension réelle.
- **width**, **height** où l'on peut indiquer une largeur et une hauteur pour la page, ou la valeur « fit » pour que la largeur et la hauteur soient celles requises par le texte qui est inclus dans l'environnement.
- **frame** qui est « off » par défaut mais peut prendre la valeur « on » si nous voulons une bordure autour du texte de la page.

5.2 Éléments sur la page

ConTeXt identifie différents zones sur les pages, dont les dimensions peuvent être configurées avec `\setuplayout`.

Nous pouvons voir toutes ces zones dans [image 5.1](#) avec les noms donnés aux différentes mesures correspondantes, et des flèches indiquant leur étendue. L'épaisseur des flèches ainsi que la taille des noms des flèches sont destinées à refléter l'importance de chacune de ces distances pour la mise en page. Si nous regardons attentivement, nous verrons que cette image montre qu'une page peut être représentée comme un tableau de 9 lignes et 9 colonnes, ou, si nous ne tenons pas compte des valeurs de séparation entre les différentes zones, il y aurait cinq lignes et cinq colonnes dont il ne peut y avoir de texte que dans trois lignes et trois colonnes. L'intersection de la ligne et de la colonne du milieu constitue la zone de texte principale et occupera normalement la majeure partie de la page.

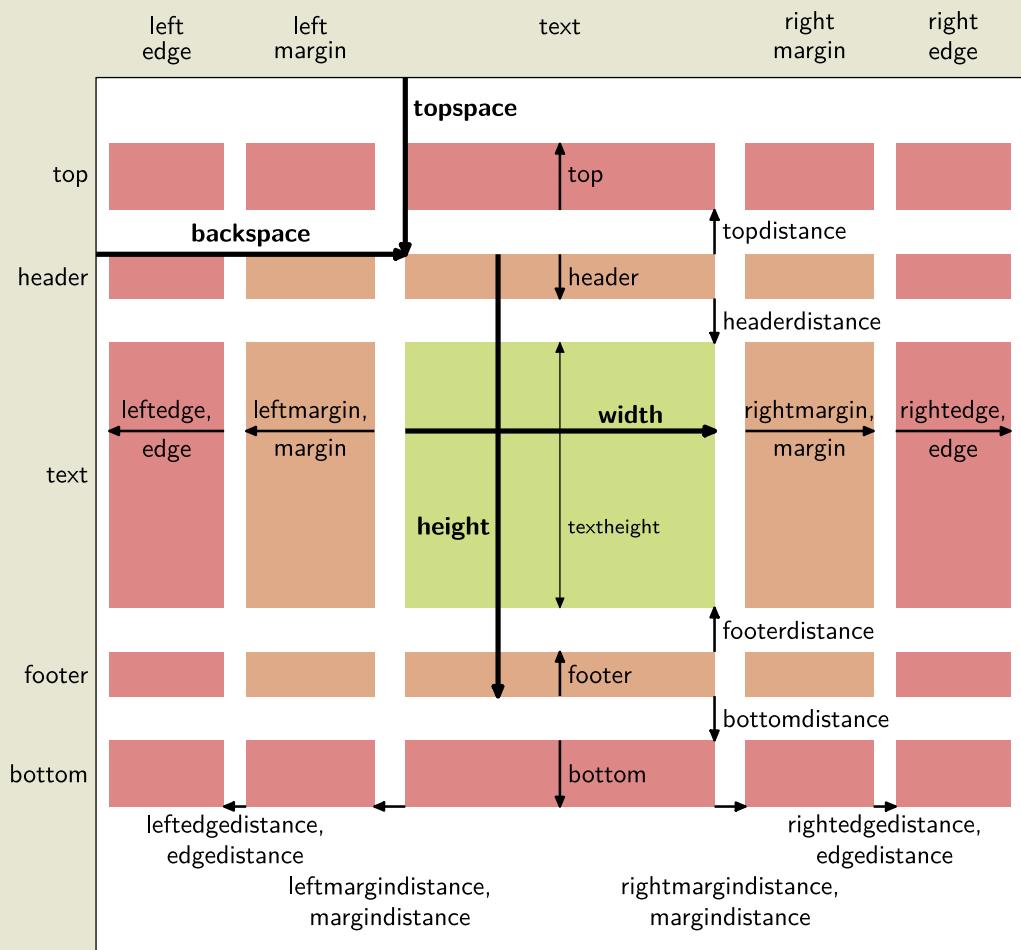


Figure 5.1 Zones et mesures sur une page

Nous allons décrire chacun des éléments de la page, en indiquant le nom à indiquer pour `\setuplayout` pour chacun d'eux :

- **Edges (Bords)** : espace blanc entourant la zone de texte. Bien que la plupart des traitements de texte les appellent « margins », il est préférable d'utiliser la terminologie de ConTeXt car elle nous permet de faire la différence entre les bords en tant que tels, où il n'y a pas de texte (bien que dans les documents électroniques, il puisse y avoir des boutons de navigation et autres), et les marges où peuvent parfois se trouver certains éléments de texte, comme, par exemple, les notes de marge.
 - La hauteur du bord supérieur est contrôlée par deux mesures : le bord supérieur lui-même (« `top` ») et la distance entre le bord supérieur et l'en-tête (« `topdistance` »). La somme de ces deux mesures est appelée « `topspace` ».
 - La taille des bords gauche et droit dépend de la « `leftedge` » « `rightedge` » respectivement. Si l'on souhaite que les deux soient de la même longueur, on peut les configurer simultanément avec l'option « `edge` ».
Dans les documents destinés à être imprimés en recto-verso, on ne parle pas de bords gauche et droit mais de bords intérieur (« `inner` ») et extérieur (« `outer` ») ; le premier est le bord le plus proche du point où les feuilles seront agrafées ou cousues, c'est-à-dire le bord gauche sur les pages impaires et le bord droit sur les pages paires. Le bord extérieur est l'opposé du bord intérieur.
 - La hauteur du bord inférieur est appelée « `bottom` ».
- **Marges (Margins)** : correctement appelé ainsi. ConTeXt appelle uniquement les marges latérales (gauche et droite) des marges. Les marges sont situées entre le bord et la zone de texte principale et sont destinées à accueillir certains éléments de texte comme, par exemple, les notes de marge, les titres de section ou leurs numéros.
Les dimensions qui contrôlent la taille des marges sont :
 - `margin` : utilisé lorsque l'on souhaite définir simultanément les marges à la même taille.
 - `leftmargin`, `rightmargin` : définit la taille des marges gauche et droite respectivement.
 - `edgedistance` : Distance entre le bord et la marge.
 - `leftgedistance`, `rightgedistance` : Distance entre le bord et les marges gauche et droite respectivement.
 - `margindistance` : Distance entre la marge et la zone de texte principale.
 - `leftmargindistance`, `rightmargindistance` : Distance entre la zone de texte principal et les marges droite et gauche respectivement.
 - `backspace` : cette mesure représente l'espace entre le coin gauche du papier et le début de la zone de texte principale. Elle est donc constituée de la somme de « `leftedge` » + « `leftgedistance` » + « `leftmargin` » + « `leftmargindistance` ».
- **En-tête (Header) et pied de page (footer)** : Ils contiennent généralement des informations qui permettent de contextualiser le texte, comme le numéro de page, le nom de l'auteur, le titre de l'ouvrage, le titre du chapitre ou de la section, etc. Dans ConTeXt ces zones de la page sont affectées par les dimensions suivantes :
 - `header` : Hauteur de l'en-tête.
 - `footer` : Hauteur du pied de page

- **headerdistance** : Distance entre l'en-tête et la zone de texte principale de la page.
- **footerdistance** : Distance entre le pied de page et la zone de texte principale de la page.
- **topdistance**, **bottomdistance** : Ces deux éléments ont été mentionnés précédemment. Il s'agit de la distance entre le bord supérieur et l'en-tête ou le bord inférieur et le pied de page, respectivement.
- **Zone de texte principal** : Il s'agit de la zone la plus large de la page, contenant le texte du document. Sa taille dépend des variables « width » et « textheight ». La variable « height », quant à elle, mesure la somme de « header », « headerdistance », « textheight », « footerdistance » et « footer ».

Dans la phase de mise en page de notre document, afin de visualiser les principaux contours de la distribution du texte, nous pouvons utiliser `\showframe` ; pour lister les valeurs des différentes mesures il faut utiliser la commande `\showsetups` ; et avec `\showlayouts` nous obtenons une combinaison des deux commandes précédentes.

5.3 Mise en page (`\setuplayout`)

5.3.1 Attribution d'une dimension aux différents composants de la page

La conception de la page implique l'attribution de tailles spécifiques aux zones respectives de la page. Cela se fait avec la commande `\setuplayout`. Cette commande nous permet de modifier n'importe laquelle des dimensions mentionnées dans la section précédente. Sa syntaxe est la suivante :

```
\setuplayout [Name] [Options]
```

où *Nom* est un argument facultatif utilisé uniquement dans le cas où nous avons conçu plusieurs dispositions (voir [section 5.3.3](#)), et les options sont, en plus d'autres que nous verrons plus tard, n'importe laquelle des mesures mentionnées précédemment. N'oubliez pas, cependant, que ces mesures sont liées entre elles puisque la somme totale des composants affectant la largeur et de ceux affectant la hauteur doit coïncider avec la largeur et la hauteur de la page. En principe, cela signifie que lorsque nous modifions une longueur horizontale, nous devons ajuster les autres longueurs horizontales, et il en va de même lorsque nous ajustons une longueur verticale.

Par défaut, ConTeXt n'effectue des ajustements automatiques des dimensions que dans certains cas qui, par ailleurs, ne sont pas indiqués de manière complète ou systématique dans sa documentation. En effectuant plusieurs tests, j'ai pu vérifier, par exemple, qu'une augmentation ou une diminution manuelle de la hauteur de l'en-tête ou du pied de page entraîne un ajustement de « `hauteur du texte` » ; en revanche, une modification manuelle de certaines marges n'ajuste pas automatiquement (selon mes tests) la largeur du texte (« `largeur` »). C'est pourquoi le moyen le plus efficace pour ne pas générer d'incohérence entre la taille de la page (définie avec `\setuppapersize`) et la taille de ses composants respectifs, est de procéder ainsi :

- En ce qui concerne les mesures horizontales :
 - En ajustant « `backspace` » (ce qui inclut « `leftedge` » et « `leftmargin` »).
 - En ajustant « `largeur` » (largeur du texte) non pas avec une dimension mais avec les valeurs « `fit` » ou « `middle` » :
 - * `fit` calcule la largeur du texte sur la base de la largeur du reste des composants horizontaux de la page.
 - * `middle` fait la même chose, mais rend d'abord les marges de droite et de gauche égales.
- En ce qui concerne les mesures verticales :

- En ajustant « `topspace` ».
- En donnant les valeurs de « `fit` » ou « `middle` » à « `height` ». Ces valeurs fonctionnent de la même manière que dans le cas de la largeur. Le premier calcule la hauteur en fonction du reste des composants, et le la seconde rend les marges supérieure et inférieure égales, puis calcule la hauteur du texte.
- Une fois que « `height` » est ajusté, en ajustant la hauteur de l'en-tête ou du pied de page si nécessaire, sachant que dans ce cas « `hauteur du texte` » sera automatiquement réajustée.
- Une autre possibilité pour déterminer de manière indirecte la hauteur de la zone de texte principale, en indiquant le nombre de lignes qu'elle doit contenir (en tenant compte de l'espace interligne et de la taille de police actuels). C'est pourquoi `\setuplayout` inclut l'option « `lines` ».

Placer la page logique sur la page physique

Dans le cas où la taille de la page logique n'est pas la même que celle de la page physique (voir section 5.1.1), `\setuplayout` nous permet de configurer certaines options supplémentaires affectant le placement de la page logique sur la page physique :

- **`location`** : Cette option détermine l'endroit où la page sera placée sur la page physique. Ses valeurs possibles sont : `left`, `middle`, `right`, `top`, `bottom`, `singlesided`, `doublesided` or `duplex`.
- **`scale`** : Indique un facteur d'échelle pour la page avant de la placer sur la page physique.
- **`marking`** : Imprime des marques visuelles sur la page pour indiquer où le papier doit être coupé.
- **`horoffset`, `veroffset`, `clipoffset`, `cropoffset`, `trimoffset`, `bleedoffset`, `artoffset`** : Une série de mesures indiquant différents déplacements dans la page physique. La plupart d'entre elles sont expliquées dans le [manuel de référence 2013](#)

Ces options `\setuplayout` doivent être combinées avec les indications de `\setuparranging` qui indique comment les pages logiques doivent être ordonnées sur la feuille de papier physique. Je n'expliquerai pas ces commandes dans cette introduction, car je n'ai pas effectué de tests à leur sujet.

Obtenir la largeur et de la hauteur de la zone de texte

Les commandes `\textwidth` et `\textheight` renvoient respectivement la largeur et la hauteur de la zone de texte. Les valeurs affichées par ces commandes ne peuvent pas être directement affichées dans le document final, mais elles peuvent être utilisées par d'autres commandes pour définir leurs mesures de largeur ou de hauteur. Ainsi, par exemple, pour indiquer que nous voulons une image dont la largeur sera de 60% de la largeur de la ligne, nous devons indiquer comme valeur de l'option « `width` » de l'image : « `width=0.6\textwidth` ».

5.3.2 Adapter la mise en page

Il se peut que notre mise en page sur une page particulière produise un résultat non souhaité ; comme, par exemple, la dernière page d'un chapitre avec seulement une ou deux lignes, ce qui n'est ni typographiquement ni esthétiquement souhaitable. Pour résoudre ces cas, ConTeXt fournit la commande `\adaptlayout` qui nous permet de modifier la taille de la zone de texte sur une ou plusieurs pages. Cette commande est destinée à être utilisée uniquement lorsque nous avons déjà terminé la rédaction de notre document et que nous procédons à quelques petits ajustements finaux. Par conséquent, son emplacement naturel est dans le préambule du document. La syntaxe de la commande est la suivante :

```
\adaptlayout [Pages] [Options]
```

où *Pages* fait référence au numéro de la page ou des pages dont on veut modifier la mise en page. Il s'agit d'un argument facultatif qui ne doit être utilisé que lorsque `\adaptlayout` est placé dans le préambule. Nous pouvons indiquer une seule page, ou plusieurs pages, en séparant les numéros par des virgules. Si nous omettons ce premier argument, `\adaptlayout` affectera exclusivement la page sur laquelle il trouve la commande.

Quant aux options, elles peuvent être :

- **height** : Permet d'indiquer, sous forme de dimensions, la hauteur que doit avoir la page en question. Nous pouvons indiquer une hauteur absolue (par exemple, "19cm") ou une hauteur relative (par exemple, "+1cm", "-0,7cm").
- **lines** : On peut inclure le nombre de lignes à ajouter ou à soustraire. Pour ajouter des lignes, la valeur est précédée d'un +, et pour soustraire des lignes, du signe -.

Considérez que lorsque nous modifions le nombre de lignes d'une page, cela peut affecter la pagination du reste du document. C'est pourquoi il est recommandé d'utiliser `\adaptlayout` uniquement à la fin, lorsque le document ne subira plus de modifications, et de le faire dans le préambule. Ensuite, nous allons à la première page que nous souhaitons adapter, nous le faisons et nous vérifions comment cela affecte les pages qui suivent ; si cela l'affecte de telle manière qu'une autre page doit être adaptée, nous ajoutons son numéro et nous compilons à nouveau, et ainsi de suite.

5.3.3 Utilisation de différentes mises en page

Si nous devons utiliser différentes mises en page dans différentes parties du document, le mieux est de commencer par définir la mise en page *générale*, puis les différentes mises en page alternatives, celles qui ne modifient que les dimensions qui doivent être différentes. Ces mises en page alternatives hériteront de toutes les

caractéristiques de la mise en page générale dont la définition ne sera pas modifiée. Pour spécifier une disposition alternative et lui donner un nom avec lequel nous pourrons l'appeler plus tard, nous utilisons la commande `\definelayout` dont la syntaxe générale est :

```
\definelayout [NomMiseEnPage/Numéro] [Configuration]
```

où *Nom/Numéro* est le nom associé à la nouvelle mise en page, ou le numéro de page où la nouvelle mise en page sera automatiquement activée, et *Configuration* contiendra les aspects de la mise en page que nous souhaitons modifier par rapport à la mise en page générale.

Lorsque la nouvelle mise en page est associée à un nom, pour l'appeler à un point particulier du document, nous utilisons :

```
\setuplayout [NomMiseEnPage]
```

et pour revenir à la disposition générale :

```
\setuplayout [reset]
```

Si, par contre, la nouvelle mise en page a été associée à un numéro de page spécifique, elle sera automatiquement activée lorsque la page sera atteinte. Cependant, une fois activée, pour revenir à la mise en page générale, nous devrons l'indiquer explicitement, même si nous pouvons le faire de manière *semi-automatique*. Par exemple, si nous voulons appliquer une mise en page exclusivement aux pages 1 et 2, nous pouvons écrire dans le préambule du document :

```
\definelayout[1][...]
\definelayout[3][reset]
```

L'effet de ces commandes sera que la mise en page définie dans la première ligne est activée sur la page 1 et sur la page 3 une autre mise en page est activée dont la fonction est uniquement de revenir à la mise en page générale.

Avec `\definelayout[even]` nous créons une mise en page qui sera activée sur toutes les pages paires ; et avec `\definelayout[odd]` la mise en page sera appliquée à toutes les pages impaires.

5.3.4 Autres questions relatives à la mise en page

A. Distinction entre les pages paires et impaires

Dans les documents imprimés recto-verso, il arrive souvent que l'en-tête, la numérotation des pages et les marges latérales diffèrent entre les pages paires (even) et impaires (odd). Les pages paires (even) sont également appelées pages de gauche (verso) et les pages impaires (odd), pages de droite (recto). Dans ces cas, il est également habituel que la terminologie concernant les marges change, et l'on parle de

marges intérieures (inner) et extérieures (outer). La première est située au point le plus proche de l'endroit où les pages seront cousues ou agrafées et la seconde sur le côté opposé. Sur les pages impaires, la marge intérieure correspond à la marge de gauche et sur les pages paires, la marge intérieure correspond à la marge de droite.

\setuplayout ne dispose d'aucune option nous permettant expressément de lui indiquer que nous voulons différencier la mise en page pour les pages impaires et paires. En effet, pour ConTeXt la différence entre les deux types de pages est définie par une option différente : \setuppagenumbering que nous verrons dans section 5.4. Une fois cette option définie, ConTeXt suppose que la page décrite avec \setuplayout était la page impaire, et construit la page paire en lui appliquant les valeurs inversées de la page impaire : les spécifications applicables sur la page impaire s'appliquent à la gauche, sur la page paire elles s'appliquent à la droite ; et vice versa : celles applicables sur la page impaire à droite, s'appliquent à la page paire à gauche.

B. Pages avec plus d'une colonne

Avec \setuplayout, nous pouvons également voir que le texte de notre document est réparti sur deux ou plusieurs colonnes, à la manière des journaux et de certains magazines, par exemple. Cette répartition est contrôlée par l'option « columns », dont la valeur doit être un nombre entier. Lorsqu'il y a plus d'une colonne, la distance entre les colonnes est indiquée par l'option « columndistance ».

Cette option est destinée aux documents dans lesquels tout le texte (ou la majeure partie du texte) est réparti sur plusieurs colonnes. Si, dans un document principalement à une colonne, nous souhaitons qu'une partie particulière soit sur deux ou trois colonnes, nous n'avons pas besoin de modifier la mise en page mais simplement d'utiliser l'environnement « columns » (voir section 12.2).

5.4 Numérotation des pages

Par défaut, ConTeXt utilise les chiffres arabes pour la numérotation des pages et le numéro apparaît centré dans l'en-tête. Pour modifier ces caractéristiques, ConTeXt dispose de différentes procédures qui, à mon avis, le rendent inutilement complexe en la matière.

Tout d'abord, les caractéristiques fondamentales de la numérotation sont contrôlées par deux commandes différentes :

`\setuppagenumbering` et `\setupuserpagenumber`.

`\setuppagenumbering` permet les options suivantes :

- **alternative** : Cette option contrôle si le document est conçu de manière à ce que l'en-tête et le pied de page soient identiques sur toutes les pages (« `singlesided` »), ou s'ils diffèrent les pages paires et impaires (« `doublesided` »). Lorsque cette option prend cette dernière valeur, les valeurs de mise en page introduites par « `setuplayout` » sont automatiquement affectées, de sorte qu'il est supposé que ce qui est indiqué dans « `setuplayout` » se réfère uniquement aux pages impaires qui sont à droite, et donc que ce qui est indiqué pour la marge de gauche se réfère en fait à la marge intérieure (qui deviendra en miroir, sur les pages paires, la marge de droite) et que ce qui est indiquée pour le côté droit se réfère en fait à la marge extérieure, qui, sur les pages paires, sera à gauche.
- **state** : Indique si le numéro de page sera affiché ou non. Il admet deux valeurs : `start` (le numéro de page sera affiché) et `stop` (les numéros de page seront supprimés). Le nom de ces valeurs (`start` et `stop`) pourrait nous faire penser que lorsque nous avons « `state=stop` » les pages cessent d'être numérotées, et que lorsque « `state=start` » la numérotation recommence. Mais ce n'est pas le cas : ces valeurs n'affectent que l'affichage ou non du numéro de page.
- **location** : indique où il sera affiché. Normalement, nous devons indiquer deux valeurs dans cette option, séparées par une virgule. Tout d'abord, nous devons préciser si nous voulons que le numéro de page figure dans l'en-tête (« `header` ») ou dans le pied de page (« `footer` »), et ensuite, à quel endroit dans l'en-tête ou le pied de page : il peut s'agir de « `left` », « `middle` », « `right` », « `inleft` », « `inright` », « `margin` », « `inmargin` », « `atmargin` » or « `marginside` ». Par exemple, pour afficher une numérotation alignée à droite dans le pied de page, il faut indiquer « `location={footer,right}` ». Voyez, d'autre part, comment nous avons entouré cette option de crochets afin que ConTeXt puisse interpréter correctement la virgule de séparation.
- **style** : indique la taille et le style de police à utiliser pour les numéros de page.
- **color** : indique la couleur à appliquer au numéro de page.
- **left** : indique le texte à ajouter à gauche du numéro de page (par exemple un tiret).
- **right** : indique le texte à ajouter à droite du numéro de page.
- **command** : indique une commande à laquelle le numéro de page sera passé en paramètre.

- **width** : indique la largeur occupée par le numéro de page.
- **strut** : Je n'en suis pas si sûr. Dans mes tests, lorsque « **strut=no** », le numéro est imprimé exactement sur le bord supérieur de l'en-tête ou sur le bas du pied de page, alors que lorsque « **strut=yes** » (valeur par défaut), un espace est appliqué entre le numéro et le bord.

\setupuserpagenumber permet les options suivantes :

- **numberconversion** : contrôle le type de numérotation qui peut être arabe (« **n** », « **numbers** »), minuscule (« **a** », « **characters** »), majuscule (« **A** », « **Characters** »), petites majuscules (« **KA** »), minuscules romaines (« **i** », « **r** », « **romannumerals** »), majuscules romaines (« **I** », « **R** », « **Roman-numerals** ») ou petites majuscules romaines (« **KR** »).
- **numéro** : indique le numéro à attribuer à la première page, sur la base duquel le reste sera calculé.
- **numberorder** : si on lui attribue la valeur « **reverse** », la numérotation des pages se fera dans l'ordre décroissant ; cela signifie que la dernière page sera la 1, l'avant-dernière la 2, etc.
- **way** : permet d'indiquer comment la numérotation va se dérouler. Cela peut être : par bloc, par chapitre, par section, par sous-section, etc.
- **prefix** : permet d'indiquer un préfixe aux numéros de page s'il ont souhaite rajouter un prefix à la numérotation (yes ou no).
- **prefixset** : permet d'indiquer le préfixe que l'on souhaite rajouter au numéro de page (par exemple « **section** » pour préfixer le numéro de page par le numéro de la section en cours).
- **numberconversionset** : Expliqué dans ce qui suit.

En plus de ces deux commandes, il faut également prendre en compte le contrôle des numéros impliquant la macrostructure du document (voir [section 7.6](#)). De ce point de vue, **\defineconversionset** permet d'indiquer un type de numérotation différent pour chacun des blocs de macro-structure. Par exemple :

```
\defineconversionset
[frontpart:pagenumber] [] [romannumerals]

\defineconversionset
[bodypart:pagenumber] [] [numbers]

\defineconversionset
[appendixpart:pagenumber] [] [Characters]
```

Vous verrez que le premier bloc de notre document (le préambule, aussi appelé frontmatter en anglais) est numéroté avec des chiffres romains minuscules, le bloc central (le corps du texte, ou bodymatter en anglais) avec des chiffres arabes et les annexes (appendices en anglais) avec des lettres majuscules.

Nous pouvons utiliser les commandes suivantes pour obtenir le numéro de page :

- `\userpagenumber` : renvoie le numéro de page tel qu'il a été configuré avec `\setuppagenumbering` et avec `\setupuserpagenumber`.
- `\pagenumber` : renvoie le même numéro que la commande précédente mais toujours en numérotation arabe.
- `\realpagenumber` : renvoie le numéro réel de la page en numérotation arabes sans tenir compte des spécifications indiquées par `\setuppagenumbering` (sans saut de numérotation, en commençant de numéroter par 1, ...).

Pour obtenir le numéro de la dernière page du document, il existe trois commandes en miroir des trois précédentes. Il s'agit de : `\lastuserpagenumber`, `\lastpage-number` et `\lastrealpagenumber`.

5.5 Sauts de page forcés ou suggérés

5.5.1 La commande `\page`

L'algorithme de distribution du texte dans ConTeXt est assez complexe et repose sur une multitude de calculs et de variables internes qui indiquent au programme quel est le meilleur endroit possible pour introduire un saut de page réel du point de vue de la pertinence typographique. La commande `\page` nous permet d'influencer cet algorithme :

- a. En suggérant certains points comme étant le meilleur endroit ou le plus inapproprié pour inclure un saut de page.

- `no` : indique que l'endroit où se trouve la commande n'est pas un bon candidat pour insérer un saut de page, donc, dans la mesure du possible, le saut doit être effectué à un autre endroit du document.
- `preference` : indique à ConTeXt que l'endroit où il rencontre la commande est un *bon endroit* pour tenter un saut de page, bien qu'il ne le forcera pas.
- `bigpreference` : indique que l'endroit où il rencontre la commande est un *très bon endroit* pour tenter un saut de page, mais il ne va pas non plus jusqu'à le forcer.

Notez que ces trois options ne forcent ni n'empêchent les sauts de page, mais indiquent seulement à ConTeXt que lorsqu'il recherche le meilleur endroit pour un saut de page, il doit prendre en compte ce qui est indiqué dans cette commande. En dernier ressort, cependant, l'endroit où le saut de page aura lieu continuera à être décidé par ConTeXt.

- b. En forçant un saut de page à un certain endroit ; dans ce cas, nous pouvons également indiquer combien de sauts de page doivent être effectués ainsi que certaines caractéristiques des pages à insérer.

- `yes` : force un saut de page à cet endroit.
- `makeup` : similaire à « `yes` », mais le saut forcé est immédiate, sans placer au préalable les objets flottants dont le placement est en attente (voir [section 13.1](#)).
- `empty` : insérer une page complètement vierge dans le document.
- `even` : insérer autant de pages que nécessaire pour que la page suivante soit une page paire.
- `odd` : insérer autant de pages que nécessaire pour que la page suivante soit une page impaire.
- `left, right` : similaire aux deux options précédentes, mais applicable uniquement aux documents imprimés en recto-verso, avec des en-têtes, pieds de page ou marges différents selon que la page est paire ou impaire.
- `quadruple` : insère le nombre de pages nécessaires pour que la page suivante soit un multiple de 4.

Outre ces options qui contrôlent spécifiquement la pagination, `\page` comprend d'autres options qui affectent le fonctionnement de cette commande. En particulier l'option « `disable` » qui fait que ConTeXt ignore les commandes `\page` qu'il trouve à partir de maintenant, et l'option « `reset` » qui produit l'effet inverse, en restaurant l'efficacité des futures commandes `\page`.

5.5.2 Joindre certaines lignes ou certains paragraphes pour empêcher l'insertion d'un saut de page entre eux

Parfois, si l'on veut empêcher un saut de page entre plusieurs paragraphes, l'utilisation de la commande `\page` peut être laborieuse, car il faudrait l'écrire à chaque endroit où il est possible qu'un saut de page soit inséré. Une procédure plus simple consiste à placer les éléments que l'on souhaite conserver sur la même page dans ce que \TeX appelle une *boîte verticale*.

Au début de ce document (sur page ??), j'ai indiqué qu'en interne, tout est une *boîte* pour \TeX . La notion de boîte est fondamentale dans \TeX pour tout type d'opération *avancée*; mais sa gestion est trop complexe pour être incluse dans cette introduction. C'est pourquoi je ne fais que des références occasionnelles aux boîtes.

Les boîtes \TeX une fois créées, sont indivisibles, ce qui signifie que nous ne pouvons pas insérer un saut de page qui couperait une boîte en deux. C'est pourquoi, si nous plaçons le contenu que nous voulons conserver en un bloc dans une boîte invisible, nous évitons l'insertion d'un saut de page qui diviserait ce contenu. La commande pour ce faire est `\vbox`, dont la syntaxe est la suivante

```
\vbox{mon contenu}
```

où *mon contenu* est le texte que nous voulons conserver en un bloc.

Certains des environnements de ConTeXt placent leur contenu dans une boîte. Par exemple, « `framedtext` », donc si nous encadrons le matériel que nous voulons garder ensemble dans cet environnement et que nous voyons également que le cadre est invisible (ce que nous faisons avec l'option `frame=off`), nous obtiendrons le même résultat.

5.6 En-têtes et pieds de page

5.6.1 Commandes pour établir le contenu des en-têtes et des pieds de page

Si nous avons attribué une certaine taille à l'en-tête et au pied de page dans la mise en page, nous pouvons y inclure du texte avec les commandes `\setupheadertexts` et `\setupfootertexts`. Ces deux commandes sont similaires, la seule différence étant que la première active le contenu de l'en-tête et la seconde celui du pied de page. Elles ont toutes deux de un à cinq arguments.

1. Utilisé avec un seul argument, il contient le texte de l'en-tête ou du pied de page qui sera placé au centre de la page. Par exemple : `\setupfootertexts [page-number]` écrira le numéro de page au centre du pied de page.
2. Utilisé avec deux arguments, le contenu du premier argument sera placé sur le côté gauche de l'en-tête ou du pied de page, et celui du second argument sur le côté droit. Par exemple, `\setupheadertexts [Preface] [pagenumber]` composera un en-tête de page dans lequel le mot « *preface* » sera écrit sur le côté gauche et le numéro de page sera imprimé sur le côté droit.
3. Si nous utilisons trois arguments, le premier indiquera *la zone* dans laquelle les deux autres doivent être imprimés. Par *zone* je fais référence aux *zones* qui se répartissent horizontalement au travers de la page, mentionnées dans [section 5.2](#), autrement dit : *edge* (*bord*), *margin* (*marge*), *text* (*texte*)... Les deux autres arguments contiennent le texte à placer dans le bord, la marge de gauche et le bord, la marge de droite.

L'utiliser avec quatre ou cinq arguments est équivalent à l'utiliser avec deux ou trois arguments, dans les cas où une distinction est faite entre les pages paires et impaires, ce qui se produit, comme nous le savons, lorsque « `alternative=doublesided` » avec `\setuppagenumbers` a été défini. Dans ce cas, deux arguments possibles sont ajoutés pour refléter le contenu des côtés gauche et droit des pages paires.

Une caractéristique importante de ces deux commandes est que, lorsqu'elles sont utilisées avec deux arguments, l'en-tête ou le pied de page central précédent (s'il existait) n'est pas réécrit, ce qui nous permet d'écrire un texte différent dans chaque zone, à condition d'écrire d'abord le texte central (en appelant la commande avec un seul argument) et d'écrire ensuite les textes des deux côtés (en l'appelant à nouveau, maintenant avec deux arguments). Ainsi, par exemple, si nous écrivons les commandes suivantes

```
\setupheadertexts [and]
\setupheadertexts [Tweedledum] [Tweedledee]
```

La première commande écrira « *et* » au centre de l'en-tête et la seconde écrira « *Tweedledum* » à gauche et « *Tweedledee* » à droite, laissant la zone centrale intacte, puisqu'il n'a pas été demandé de la réécrire. L'en-tête qui en résulte se présente alors comme suit



L'explication que je viens de donner du fonctionnement de ces commandes est ma conclusion après de nombreux tests. L'explication de ces commandes fournie dans ConTeXt *excursion* est basée sur la version avec cinq arguments ; et celle du manuel de référence 2013 est basée sur la version avec trois arguments. Je pense que ma est plus claire. D'autre part, je n'ai pas vu d'explication sur la raison pour laquelle le deuxième appel de commande n'écrase pas l'appel précédent, mais c'est ainsi que cela fonctionne si nous écrivons d'abord l'élément central dans l'en-tête ou le pied de page, puis ceux de chaque côté. Mais si nous écrivons d'abord les éléments de chaque côté dans l'en-tête ou le pied de page, l'appel ultérieur à la commande d'écriture de l'élément central effacera les en-têtes ou pieds de page précédents. Pourquoi ? Je n'en ai aucune idée. Je pense que ces petits détails introduisent une complication inutile et devraient être clairement expliqués dans la documentation officielle.

En outre, nous pouvons indiquer toute combinaison de texte et de commandes comme contenu de l'en-tête ou du pied de page. Mais aussi les valeurs suivantes :

- **date, currentdate** : écrira (l'un ou l'autre) la date du jour.
- **pagenumber** : écrira le numéro de page.
- **part, chapter, section...** : écrira le titre correspondant à la partie, au chapitre, à la section... ou à toute autre division structurelle.
- **partnumber, chapternumber, sectionnumber...** : indique le numéro de la partie, du chapitre, de la section... ou de toute autre division structurelle.

Attention : Ces noms symboliques (**date, currentdate, pagenumber, chapter, chapternumber**, etc.) ne sont interprétés comme tels que si le nom symbolique lui-même est le seul contenu de l'argument ; mais si nous ajoutons une autre commande de texte ou de formatage, ces mots seront interprétés littéralement, et ainsi, par exemple, si nous écrivons **\setupheadertexts[chapternumber]** nous obtiendrons le numéro du chapitre actuel ; mais si nous écrivons **\setupheadertexts[Chapitre chapternumber]** nous obtiendrons : « Chapitre chapternumber ». Dans ces cas, lorsque le contenu de la commande n'est pas seulement le mot symbolique, nous devons :

- Pour **date, currentdate** et **pagenumber**, utilisez, non pas le mot symbolique, mais la commande du même nom (**\date, \currentdate** ou **\pagenumber**).
- Pour **part, partnumber, chapter, chapternumber**, etc., utilisez la commande **\getmarking[info]** qui renvoie le contenu de l'*info* demandée. Ainsi, par exemple, **\getmarking[chapter]** renvoie le titre du chapitre en cours, tandis que **\getmarking[chapternumber]** renvoie son numéro.

5.6.2 Mise en forme des en-têtes et des pieds de page

Le format spécifique dans lequel le texte de l'en-tête ou du pied de page est affiché peut être indiqué dans les arguments de **\setupheadertexts** ou **\setupfootertexts** en utilisant les commandes de format correspondantes dans les crochets des différents éléments.

Cependant, on peut aussi configurer cela globalement avec **\setupheader** et **\setupfooter** qui offrent les options suivantes :

- **state** : permet les valeurs suivantes : `start`, `stop`, `empty`, `high`, `none`, `normal` ou `nomarking`.
- **style**, **leftstyle**, **rightstyle** : configuration du style de texte de l'en-tête et du pied de page. `style` affecte toutes les pages, `leftstyle` les pages paires et `rightstyle` les pages impaires.
- **color**, **leftcolor**, **rightcolor** : couleur de l'en-tête ou du pied de page. Elle peut affecter toutes les pages (option `color`) ou seulement les pages paires (`leftcolor`) ou impaires (`rightcolor`).
- **width**, **leftwidth**, **rightwidth** : largeur de tous les en-têtes et pieds de page (`width`) ou des en-têtes/pieds de page sur les pages paires (`leftwidth`) ou impaires (`rightwidth`).
- **before** : commande à exécuter avant d'écrire l'en-tête ou le pied de page.
- **after** : commande à exécuter après l'écriture de l'en-tête ou du pied de page.
- **strut** : si renseigné avec « yes », ConTeXt va donner une profondeur et une hauteur en lien avec la police utilisée, même si le texte utilisé en question n'utilise que des lettres sans jambage (la patte du p, la partie supérieure d'un h...). En conséquence, un espace de séparation vertical est établi entre le texte de l'en-tête et le bord supérieur de la zone d'en-tête. Si renseigné avec « no », la hauteur et la profondeur de ligne collent strictement au texte de l'en-tête et ce dernier vient buter contre le bord supérieur de la zone d'en-tête.

Pour désactiver (voire modifier) les en-têtes et les pieds de page « localement », c'est à dire sur une page particulière, utilisez les commandes `\setupheader[state=empty]` `\setupfooter[state=empty]` au sein de la page. Ces commandes agissent exclusivement sur la page où elles se trouvent.

5.6.3 Définir des en-têtes et des pieds de page spécifiques et les relier aux commandes de section

Jusqu'à présent, le système d'en-tête et de pied de page de ConTeXt nous permet de changer automatiquement le texte de l'en-tête ou du pied de page lorsque nous changeons de chapitre ou de section, ou lorsque nous changeons de page, si nous avons défini des en-têtes ou des pieds de page différents pour les pages paires et impaires. Mais ce qu'il ne permet pas, c'est de différencier la première page (du document, d'un chapitre ou d'une section) du reste des pages. Pour réaliser ce dernier point, nous devons :

1. Définir un en-tête ou un pied de page spécifique.
2. Le lier à la section à laquelle il doit s'appliquer.

La définition d'en-têtes ou de pieds de page spécifiques s'effectue à l'aide de la fonction `\definetext`, dont la syntaxe est la suivante :

```
\definetext
  [Name] [Type]
  [Content1] [Content2] [Content3]
  [Content4] [Content5]
```

où *Name* est le nom attribué à l’en-tête ou au pied de page dont il s’agit ; *Type* peut être **header** ou **footer**, selon celui que nous définissons, et les cinq arguments restants contiennent le contenu que nous voulons pour le nouvel en-tête ou pied de page, de manière similaire à la façon dont nous avons vu les fonctions **\setupheadtexts** et **\setupfootertexts**.

Une fois cette opération effectuée, nous devons lier le nouvel en-tête ou le nouveau pied de page à une section particulière avec **\setuphead** en utilisant les options *header* et *footer* (qui ne sont pas expliquées dans [Chapter 7](#)).

Ainsi, l’exemple suivant masquera l’en-tête de la première page de chaque chapitre et un numéro de page centré apparaîtra en pied de page :

```
\setuphead  
[chapter]  
[footer=ChapterFirstPage]
```

	1	
1 Test	We thrive in information-thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeon-hole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.	



1 Test

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

1



5.7 Insertion d'éléments de texte dans les bords de page et les marges

Les bords supérieur et inférieur et les marges droite et gauche ne contiennent généralement pas de texte d'aucune sorte. Cependant, ConTeXt permet d'y placer certains éléments de texte. En particulier, les commandes suivantes sont disponibles à cet effet :

- `\setupopttexts` : permet de placer du texte sur le bord supérieur de la page (au-dessus de la zone d'en-tête).
- `\setupbottomtexts` : permet de placer du texte au bord inférieur de la page (sous la zone de pied de page).
- `\margintext, \atleftmargin, \atrightmargin, \ininner, \ininneredge, \ininnermargin, \inleft, \inleftedge, \inleftmargin, \inmargin, \inother, \inouter, \inouteredge, \inoutermargin, \inright, \inrightedge, \inrightmargin` : nous permettent de placer du texte dans les bords latéraux et les marges du document.

Les deux premières commandes fonctionnent exactement comme `\setupheadertexts` et `\setupfootertexts`, et le format de ces textes peut même être configuré à l'avance avec `\setuptop` et `\setupbottom` de la même manière que `\setupheader` nous permet de configurer les textes pour `\setupheadertexts`. Pour tout cela, je renvoie à ce que j'ai déjà dit dans [section 5.6](#). Le seul petit détail à ajouter est que le texte mis en place pour `\setupopttexts` ou `\setupbottomtexts` ne sera pas visible si aucun espace n'a été réservé dans la mise en page pour les bords supérieur (top) ou inférieur (bottom). Pour cela, voir [section 5.3.1](#).

Quant aux commandes visant à placer du texte dans les marges du document, elles ont toutes une syntaxe similaire car sont toutes définies par `\definemargin` avec des options particulières pour chacunes.

`\CommandName [Reference] [Configuration] {Text}`

où *Reference* et *Configuration* sont des arguments optionnels ; le premier est utilisé pour d'éventuelles références croisées et le second nous permet de configurer le texte de la marge. Le dernier argument, entre crochets, contient le texte à placer dans la marge.

exemple de
`\inouter`, où l'on constate que
le texte est dans la marge extérieure et justifié vers l'interne

Parmi ces commandes, une habituellement utilisée est `\margintext` car elle permet de placer le texte dans la marge de gauche de la page avec une justification à droite. Les autres commandes, comme leur nom l'indique, placent le texte dans la marge elle-même (droite ou gauche, intérieure ou extérieure), ou dans le bord (droit ou gauche, intérieur ou extérieur). Ces commandes sont étroitement liées à la mise en page car si, par exemple, nous utilisons `\inrightedge` mais que nous n'avons pas réservé d'espace dans la mise en page pour le bord droit, rien ne sera visible.

Les options de configuration de `\margintext` sont les suivantes :

- **location** : indique dans quelle marge le texte sera placé. Elle peut être `left`, `right` ou, dans les documents recto-verso, `outer` ou `inner`. Par défaut, il s'agit de `left` pour les documents recto et de `outer` pour les documents recto-verso.
- **width** : largeur disponible pour l'impression du texte. Par défaut, la largeur totale de la marge sera utilisée.
- **margin** : indique si le texte sera placé dans la `margin` elle-même ou dans le `edge`, voire directement au contact du texte principal avec `normal`.
- **align** : alignement du texte. Les mêmes valeurs sont utilisées ici que dans [\setupalign 11.6.1](#).
- **line** : permet d'indiquer un nombre de lignes de déplacement du texte dans la marge. Ainsi, `line=1` déplacera le texte d'une ligne en dessous et `line=-1` d'une ligne au-dessus.
- **style** : commande ou commandes permettant d'indiquer le style du texte à placer dans les marges.
- **color** : couleur du texte des marges.
- **command** : nom d'une commande à laquelle le texte à placer dans la marge sera passé en argument. Cette commande sera exécutée avant d'écrire le texte. Par exemple, si nous voulons dessiner un cadre autour du texte, nous pouvons utiliser « `[command={\framed}]` ».

Les autres commandes offrent les mêmes options, à l'exception de `location` et `margin`. En particulier, les commandes `\atrightmargin` et `\atleftmargin` placent le texte complètement collé au corps de la page. Nous pouvons établir un espace de séparation avec l'option `distance`, que je n'ai pas mentionnée en parlant de `\margintext`.

En plus des options ci-dessus, ces commandes prennent également en charge d'autres options (`strut`, `anchor`, `method`, `category`, `scope`, `option`, `hoffset`, `voffset`, `dy`, `bottomspace`, `threshold` et `stack`) que je n'ai pas mentionnées parce qu'elles ne sont pas documentées et que, franchement, je ne suis pas très sûr de leur utilité. Ceux qui ont des noms comme `distance`, on peut les deviner, mais le reste ? Le wiki ne mentionne que l'option `stack`, disant qu'elle est utilisée pour émuler la commande `\marginpars` de L^AT_EX, mais cela ne me semble pas très clair.

```
\setupmargindata [...]1 [...] [...] 2 [...]
OPT
1 NAME
2 strut      = yes no auto cap fit line default CHARACTER
  command    = '\...##1
  width      = DIMENSION
  align      = inherits: \setupalign
  anchor     = region text
  location   = left right inner outer
  method     = top line first depth height
  category   = default edge
  scope      = local global
  option     = text paragraph
  margin     = local normal margin edge
  distance   = DIMENSION
  hoffset    = DIMENSION
  voffset    = DIMENSION
  dy         = DIMENSION
  bottomspace = DIMENSION
  threshold  = DIMENSION
  line       = NUMBER
  stack      = yes continue
  style      = STYLE COMMAND
  color      = COLOR
```

La commande `\setupmargindata` nous permet de configurer globalement les textes de chaque marge. Ainsi, par exemple,

```
\setupmargindata[right][style=slanted]
```

fera en sorte que tous les textes de la marge de droite soient écrits en style oblique.

Nous pouvons également créer notre propre commande personnalisée avec

```
\definemargindata[Name][Configuration]
```

Chapitre 6

Polices d'écriture et couleurs dans ConTEXt

Table of Contents : 6.1 Polices de caractères incluses dans « ConTEXt Standalone » ; 6.2 Caractéristiques d'une police ; 6.2.1 Polices, styles et styles alternatifs ; A Style de police ; B Styles alternatifs ; C Différence entre l'italique et l'oblique ; 6.2.2 Taille de police ; 6.3 Définition de la police principale du document ; 6.4 Modification de la police ou de certaines de ses caractéristiques ; 6.4.1 Les commandes \setupbodyfont et \switchtobodyfont ; 6.4.2 Changement rapide de style, d'alternative et de taille ; A Changement de style et de style alternatif ; B Commandes pour changer d'alternative et de taille en même temps ; C Personnalisation des facteurs d'échelle et des suffixes ; 6.4.3 Définition de commandes et de mots clés pour les tailles, les styles et les styles alternatifs de polices ; 6.5 Autres questions relatives à l'utilisation de styles alternatifs ; 6.5.1 Italique, oblique et mise en valeur ; 6.5.2 Petites majuscules et fausses petites majuscules ; 6.6 Utilisation et configuration des couleurs ; 6.6.1 Utiliser des couleurs pour des éléments textuels ; 6.6.2 Utiliser des couleurs en arrière-plan et pour le texte en général ; 6.6.3 Utiliser des couleurs pour des portions de texte ; 6.6.4 Couleurs prédéfinies ; 6.6.5 Visualiser les couleurs disponibles ; 6.6.6 Définir ses propres couleurs ; 6.7 Bonus 1 - Utilisation des polices du système d'exploitation ; 6.7.1 Emplacement des polices sur votre ordinateur ; 6.7.2 Utilisation rapide d'une nouvelle police de caractères ; 6.7.3 Utilisation de divers styles alternatifs de police ; 6.7.4 Installation d'un typecript pour l'utiliser partout ; 6.7.5 Quelques dernières fonctionnalités avec les polices ;

6.1 Polices de caractères incluses dans « ConTEXt Standalone »

Le système de polices de ConTEXt offre de nombreuses possibilités, mais il est également assez complexe. Je n'analyserai pas toutes les possibilités avancées de polices dans ce manuel, mais je me limiterai à supposer que nous travaillons avec certaines des 21 polices fournies avec l'installation de ConTEXt Standalone, celles présentées dans [table 6.1](#).

La colonne centrale de [table 6.1](#) indique le ou les noms par lesquels ConTEXt connaît la police en question. Lorsqu'il y a deux noms, ils sont synonymes. La dernière colonne présente un exemple de la police utilisée. Quant à l'ordre dans lequel les polices sont présentées, la première est la police que ConTEXt utilise par défaut, les

Nom official	Référence ConTeXt	Exemple
Latin Modern	modern, modern-base	Emily Brontë's book
Latin Modern Variable	modernvariable, modern-variable	Emily Brontë's book
TeX Gyre Adventor	adventor, avantgarde	Emily Brontë's book
TeX Gyre Bonum	bonum, bookman	Emily Brontë's book
TeX Gyre Cursor	cursor, courier	Emily Brontë's book
TeX Gyre Heros	heros, helvetica	Emily Brontë's book
TeX Gyre Schola	schola, schoolbook	Emily Brontë's book
Tex Gyre Chorus	chorus, chancery	<i>Emily Brontë's book</i>
Tex Gyre Pagella	pagella, palatino	Emily Brontë's book
Tex Gyre Termes	termes, times	Emily Brontë's book
DejaVu	dejavu dejavu-condensed	Emily Brontë's book Emily Brontë's book
Gentium	gentium	Emily Brontë's book
Antykwa Poltawskiego	antykwapoltawskiego	Emily Brontë's book
Antykwa Toruńska	antykwatorunska	Emily Brontë's book
Iwona	iwona	Emily Brontë's book
Kurier	kurier	Emily Brontë's book
PostScript	postscript	Emily Brontë's book
Euler	eulernova	Emily Brontë's book
Stix2	stix	Emily Brontë's book
Xits	xits	Emily Brontë's book

Tableau 6.1 Fonts included in the ConTeXt distribution

autres polices sont classées par ordre alphabétique, tandis que les trois dernières polices sont spécifiquement conçues pour les mathématiques. Notez que la police Euler ne peut pas représenter directement les lettres accentuées, nous obtenons donc Bront's, et non Brontë's.

Pour les lecteurs venant du monde Windows et de ses polices par défaut, j'indiquerai que *heros* équivaut à Arial dans Windows, tandis que *termes* équivaut à Times New Roman. Elles ne sont pas exactement les mêmes mais suffisamment similaires, au point qu'il faudrait être très observateur pour faire la différence.

Les polices utilisées par Windows ne sont pas des *logiciels libres* (en fait, presque rien dans Windows n'est un *logiciel libre*), elles ne peuvent donc pas être incluses dans une distribution de ConTeXt. Cependant, si ConTeXt est installé sous Windows, alors ces polices sont déjà installées et peuvent être utilisées comme n'importe quelle autre police installée sur le système exécutant ConTeXt. Dans cette introduction, cependant, je ne traiterai pas de la manière d'utiliser les polices déjà installées sur le système. Vous trouverez de l'aide à ce sujet sur le wiki ConTeXt.

6.2 Caractéristiques d'une police

6.2.1 Polices, styles et styles alternatifs

La terminologie concernant les polices est quelque peu confuse, car parfois ce que l'on appelle une police est en réalité une *famille de polices* qui comprend différents styles et variantes partageant un design de base. Je n'entrerai pas dans la question de savoir quelle terminologie est la plus correcte ; je m'intéresse uniquement à la clarification de la terminologie utilisée dans ConTeXt. Ce dernier fait une distinction entre les polices, les styles et les variantes (ou alternatives) pour chaque style. Les *polices* incluses dans la distribution ConTeXt (il s'agit en fait de *familles de polices*) sont celles que nous avons vues dans la section précédente. Nous allons maintenant nous pencher sur les *styles* et les *alternatives*.

A. Style de police

DONALD E. KNUTH a conçu la police *Computer Modern* pour TeX, en lui donnant trois *styles* distincts appelés *roman*, *sans serif* et *teletype*. Le style *roman* est une conception dans laquelle les caractères présentent des « empattements » à chaque extrémité, ou « serif » dans le jargon typographique, ce qui explique pourquoi ce style de police est également connu sous le nom *serif*. Ce style était considéré comme le style normal ou par défaut. Le style *sans serif*, comme son nom l'indique, est dépourvu de ces empattements et constitue donc une police plus simple et plus stylisée, parfois connue sous d'autres noms, par exemple en français, *linéale* ; cette police peut être la police principale du document, mais elle est également appropriée pour être utilisée pour distinguer certains fragments d'un texte dont la police principale est de style *roman*, comme, par exemple, les titres ou les en-têtes de page. Enfin, la police *teletype* a été incluse dans la police *Computer Roman* car elle a été conçue pour l'écriture de livres de programmation informatique, comportant de grandes sections de code informatique qui sont conventionnellement représentées, dans les documents imprimés, dans un style monospace qui imite les terminaux informatiques et les anciennes machines à écrire. Voici une illustration :

- Style avec serif
- Style sans serif
- Style monospace

Un quatrième style destiné aux fragments de mathématiques pourrait être ajouté à ces trois styles de police. Mais comme TeX utilise automatiquement ce style lorsqu'il entre en mode mathématique, et qu'il n'inclut pas de commandes pour l'activer ou le désactiver expressément, et qu'il ne possède pas non plus les *variantes de style* ou les alternatives des autres styles, il n'est pas habituel de le considérer comme un *style* proprement dit.

ConTeXt inclut des commandes pour deux styles supplémentaires possibles : manuscrit et calligraphique. Je ne suis pas exactement sûr de la différence entre eux car, d'une part, aucune des polices incluses dans la distribution de ConTeXt inclut ces styles dans leur conception, et d'autre part, comme je le vois, l'écriture calligraphique est également manuscrite. Ces commandes que ConTeXt inclut pour activer de tels styles, si elles sont utilisées avec une police qui ne les implémente pas, ne causeront aucune erreur lors de la compilation : c'est simplement que rien ne se passe.

B. Styles alternatifs

Chaque *style* offre un certain nombre de styles alternatifs, et c'est ainsi que le ConTeXt les appelle, (*alternative*) :

- Régulier (Regular) ou normal (« `tf` », à partir de *typeface*) : *style regular*
- Gras (Bold) (« `bf` », à partir de *boldface*) : **style gras**
- Italique (Italic) (« `it` », à partir de *italic*) : *style italique*
- Gras Italique (BoldItalic) (« `bi` », à partir de *bold italic*) : **style gras italique**
- Oblique (Slanted) (« `sl` » à partir de *slanted*) : *style slanted*
- Gras Oblique (BoldSlanted) (« `bs` » à partir de *bold slanted*) : **style gras oblique**
- Petites Majuscules (Small caps) (« `sc` » à partir de *small caps*) : **STYLE SMALL CAPS**
- Médieval (Medieval) (« `os` » à partir de *old style*) : *style medieval*

Ces *alternatives*, comme leur nom l'indique, sont mutuellement exclusives : lorsque l'une d'elles est activé, les autres sont désactivés. C'est pourquoi ConTeXt fournit des commandes pour les activer mais pas pour les désactiver ; parce que lorsque nous activons une alternative, nous désaktivons celle que nous utilisions jusqu'alors ; et donc, par exemple, si nous écrivons en italique et que nous activons le gras, l'italique sera désactivé. Si nous voulons utiliser simultanément le gras et l'italique, nous ne devons pas activer l'un puis l'autre, mais activer l'alternative qui inclut les deux (« `bi` »).

D'autre part, il faut garder à l'esprit que même si ConTeXt suppose que chaque police aura ces alternatives, et fournit donc des commandes pour les activer, pour fonctionner et produire un effet perceptible dans le document final, ces commandes ont besoin que la police ait des styles spécifiques dans sa conception pour chaque style et alternative.

En particulier, de nombreuses polices ne font pas la différence dans leur conception entre les lettres inclinées et italiques, ou n'incluent pas de styles spéciaux pour les petites majuscules.

C. Différence entre l'italique et l'oblique

La similitude de la fonction typographique assurée par l'italique et l'oblique conduit de nombreuses personnes à confondre ces deux alternatives. La lettre oblique est obtenue par une légère rotation du style régulier. Mais l'italique implique – du moins dans certaines polices – une conception différente dans laquelle les lettres *semblent inclinées* parce qu'elles ont été dessinées pour y ressembler ; mais en réalité, il n'y a pas d'inclinaison authentique. C'est ce que montre l'exemple suivant, dans lequel nous avons écrit le même mot trois fois à la même taille suffisamment grande pour qu'il soit facile d'apprécier les différences. Dans la première version, le style régulier est utilisé, dans la deuxième, l'inclinaison, et dans la troisième, l'italique :

```
\definebodyfontenvironment[44pt]
\setupbodyfont[modern,44pt]
{\rm italics} --
{\sl italics} --
{\it italics}
```

italics – *italics* – *italics*

Notez que le dessin des caractères est le même dans les deux premiers exemples, mais que dans le troisième, il y a de subtiles différences dans les traits de certaines lettres, ce qui est très évident, notamment dans la façon dont le « a » est dessiné, bien que les différences se produisent en fait dans presque tous les caractères.

Les utilisations habituelles des lettres italiques et inclinées sont similaires et chaque personne décide d'utiliser l'une ou l'autre. Il y a là une liberté, même s'il faut souligner qu'un document sera mieux composé et aura un meilleur aspect si l'utilisation de l'italique et des lettres obliques est *cohérente*. De plus, dans de nombreuses polices, la différence de conception entre l'italique et l'oblique est négligeable, de sorte que l'utilisation de l'une ou de l'autre ne fait aucune différence.

D'autre part, l'italique et l'oblique sont tous deux des alternatives aux polices de caractères, ce qui signifie principalement deux choses :

1. Nous ne pouvons les utiliser que lorsqu'elles sont définies dans la police.
2. Lorsqu'on active l'une d'entre elles, on désactive l'alternative qui était utilisée jusqu'alors.

Outre les commandes d'italique et d'oblique, ConTeXt offre une commande supplémentaire pour *mettre en valeur* un texte particulier. Son utilisation implique des différences subtiles par rapport à l'italique ou à l'incliné. Voir section 6.5.1.

6.2.2 Taille de police

Toutes les polices gérées par ConTeXt sont basées sur des graphiques vectoriels, de sorte qu'en théorie elles peuvent être affichées à n'importe quelle taille de police, bien que, comme nous le verrons, cela dépende des instructions réelles que nous utilisons pour déterminer la taille de la police. Sauf indication contraire, il est supposé que la taille de la police sera de 12 points.

Toutes les polices utilisées par ConTeXt sont basées sur le graphisme vectoriel, et sont donc des polices Opentype ou Type 1, ce qui implique que les polices dont les origines sont antérieures à cette technologie ont été réimplémentées. En particulier, la police par défaut de TeX *Computer Modern*, conçue par Knuth, n'existe que dans certaines tailles, et a donc été réimplémentée dans une conception appelée *Latin Modern* utilisée par ConTeXt, bien que dans de nombreux documents, elle continue d'être appelée *Computer Modern* en raison du fort symbolisme que cette police a toujours pour les systèmes TeX, puisque ceux-ci ont été créés et développés par Knuth en même temps qu'un autre programme appelé METAFONT, destiné à concevoir des polices pouvant fonctionner avec TeX.

6.3 Définition de la police principale du document

Par défaut, sauf si une autre police est indiquée, ConTeXt utilisera *Latin Modern Roman* à 12 points comme police principale. Cette police a été conçue à l'origine par KNUTH pour être implémentée dans TeX. Il s'agit d'une police élégante de style romain avec de grandes variations d'épaisseur et des empattements – appelées *serifs* – dans certains traits, ce qui est très approprié à la fois pour les textes imprimés et pour l'affichage à l'écran ; cependant – et c'est une opinion personnelle – elle n'est pas si adaptée aux petits écrans comme le *smartphone*, parce que les *serifs* ou les floriatures ont tendance à s'empiler, rendant la lecture difficile.

Pour configurer une police différente, nous utilisons `\setupbodyfont` qui nous permet non seulement de changer la police actuelle, mais aussi sa taille et son style. Si nous voulons que cette option s'applique à l'ensemble du document, nous devons l'inclure dans le préambule du fichier source. Mais si nous souhaitons simplement changer la police à un moment donné, c'est ici que nous devons inclure ce qui suit.

Le format `\setupbodyfont` est le suivant :

```
\setupbodyfont [Options]
```

```
\setupbodyfont [....,*...]
               OPT
*  DIMENSION NAME global reset x xx small big script scripts script rm ss tt hw cg roman serif
   regular sans sansserif support type teletype mono handwritten calligraphic
```

où les différentes options de la commande nous permettent d'indiquer :

- **Le nom de la police**, qui peut être n'importe lequel des noms de police symboliques trouvés dans [table 6.1](#).
- **La taille**, qui peut être indiquée soit par ses dimensions (en utilisant le point comme unité de mesure), soit par certains noms symboliques. Mais notez que même si j'ai dit précédemment que les polices utilisées par ConTeXt peuvent être mises à l'échelle à pratiquement n'importe quelle taille, dans `\setupbodyfont`, seules les tailles constituées de nombres entiers compris entre 4 et 12, ainsi que les valeurs 14,4 et 17,3, sont prises en charge par ConTeXt. Par défaut, il suppose que la taille est de 12 points.
`\setupbodyfont`, établit ce que l'on pourrait appeler la taille de base du document, c'est-à-dire la taille de caractère normale sur la base de laquelle sont calculées les autres tailles, par exemple les titres et les notes de bas de page. Lorsque nous modifions le corps principal avec `\setupbodyfont`, tous les autres corps calculés sur la base de la police principale sont également modifiés.

En plus d'indiquer directement le corps de caractère (10pt, 11pt, 12pt, etc.), nous pouvons également utiliser certains noms symboliques qui calculent le corps de caractère à appliquer, sur la base du corps actuel. Les noms symboliques en question sont, du plus grand au plus petit : big, small, script, x, scriptscript et xx. Ainsi, par exemple, si nous voulons définir un corps de texte avec `\setupbodyfont` dont la taille est supérieure à 12 points, nous pouvons le faire avec « big ».

Pour utiliser une taille de police différentes de celles disponibles par défaut, il est nécessaire de la déclarer avec `\definebodyfontenvironment` préalablement à l'utilisation de `\setupbodyfont`.

```
\setupbodyfont [modern,17.3pt]
Coucou
```

Coucou

```
\setupbodyfont [modern,17.5pt]
Coucou
```

Coucou

```
\definebodyfontenvironment [17.5pt]
\setupbodyfont [modern,17.5pt]
Coucou
```

Coucou

- le **style de police**, qui, comme nous l'avons indiqué, peut être romain (avec empattements), ou sans empattements (sans serif), ou style machine à écrire, et pour certaines polices, style manuscrit et calligraphique. `\setupbodyfont` autorise différents noms symboliques pour indiquer les différents styles. Ceux-ci se trouvent dans la table 6.2 :

Style	Noms symboliques autorisés
Roman	<code>rm</code> , <code>roman</code> , <code>serif</code> , <code>regular</code>
Sans Serif	<code>ss</code> , <code>sans</code> , <code>support</code> , <code>sansserif</code>
Monospace	<code>tt</code> , <code>mono</code> , <code>type</code> , <code>teletype</code>
Manuscrite	<code>hw</code> , <code>handwritten</code>
Calligraphique	<code>cg</code> , <code>calligraphic</code>

Tableau 6.2 Styles avec `setupbodyfont`

Pour autant que je sache, les différents noms utilisés pour chacun des styles sont totalement synonymes.

Visualiser une police

Avant de décider d'utiliser une police particulière dans notre document, nous voudrions normalement voir à quoi elle ressemble. Cela peut presque toujours être fait à partir du système d'exploitation car il existe généralement un utilitaire permettant d'examiner l'apparence des polices installées sur le système ; mais pour plus de commodité, ConTeXt lui-même offre un utilitaire qui nous permet de voir l'apparence de n'importe quelle police activée dans ConTeXt. Il s'agit de `\showbodyfont`, qui génère un tableau avec des exemples de la police que nous indiquons.

Le format de `\showbodyfont` est le suivant :

```
\showbodyfont [Options]
```

où nous pouvons indiquer comme options précisément les mêmes noms symboliques que dans `\setupbodyfont`. Ainsi, l'exemple affiché nous montre différentes illustrations des polices schola et adventor avec une taille de base de 14 points.

```
\definebodyfontenvironment[14pt]
\showbodyfont[schola,14pt]
\blank[big]
\showbodyfont[adventor,14pt]
```

[schola] [schola,14pt] \mr : Ag														
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfdb	
\rm	Ag	Ag	Ag	Ag	Ag	Ag	Ag							
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag							
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag							

[adventor] [adventor,14pt] \mr : Ag														
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfdb	
\rm	Ag	Ag	Ag	Ag	Ag	Ag	Ag							
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag							
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag							

Notez qu'il y a certaines commandes dans la première ligne et la première colonne du tableau. Plus loin, lorsque la signification de ces commandes aura été expliquée, nous examinerons à nouveau les tableaux générés par `\showbodyfont`.

Si nous voulons voir la gamme complète des caractères d'une police spécifique, nous pouvons utiliser la commande `\showfont [FontName]`. Cette commande affichera le dessin principal de chacun des caractères sans appliquer les commandes de styles et d'alternatives.

```
\showfont [texgyreadventorregular]
```

6.4 Modification de la police ou de certaines de ses caractéristiques

6.4.1 Les commandes `\setupbodyfont` et `\switchtobodyfont`

Pour changer la police, le style ou la taille, nous pouvons utiliser la même commande avec laquelle nous avons établi la police au début du document, lorsque nous ne voulons pas utiliser la police par défaut de ConTeXt : `\setupbodyfont`. Il suffit de placer cette commande à l'endroit du document où l'on souhaite changer de police. Elle produira un changement de police *permanent*, ce qui signifie qu'elle affectera directement la police principale et indirectement toutes les polices qui lui sont liées.

`\switchtobodyfont` est très similaire à `\setupbodyfont`. Les deux commandes nous permettent de modifier les mêmes aspects de la police (la police elle-même, le style et la taille) mais, en interne, elles fonctionnent différemment et sont destinées à des utilisations différentes. La première (`\setupbodyfont`) sert à établir la police principale (et normalement la seule) du document ; il n'est ni courant ni correct d'un point de vue typographique qu'un document ait plus d'une police principale (c'est pourquoi elle est appelée police principale). En revanche, `\switchtobodyfont` est destiné à écrire certaines parties d'un texte dans une police différente, ou à affecter une police particulière à un type spécial de paragraphe que nous voulons définir dans notre document.

En dehors de ce qui précède – qui affecte en fait le fonctionnement interne de chacune de ces deux commandes – du point de vue de l'utilisateur, il existe quelques différences entre l'utilisation de l'une ou l'autre commande. En particulier :

1. Comme nous le savons déjà, `\setupbodyfont` est limitée à une gamme particulière de tailles, alors que `\switchtobodyfont` nous permet d'indiquer pratiquement n'importe quelle taille, de sorte que si la police n'est pas disponible dans cette taille, elle s'y adaptera.
2. `\switchtobodyfont` n'affecte pas les éléments textuels autrement que là où il est utilisé, contrairement à `\setupbodyfont` qui, comme mentionné ci-dessus, établit la police principale et, en la modifiant, modifie également la police de tous les éléments textuels dont la police est calculée sur la base de la police principale.

Ces deux commandes, en revanche, modifient non seulement la police, le style et la taille, mais aussi d'autres aspects associés à la police comme, par exemple, l'interligne.

```
\setupbodyfont[modern]
Coucou
\switchtobodyfont[17.5pt]
Coucou
```

Coucou Coucou

```
\setupbodyfont [modern,17.5pt]
Coucou
% 17.5pt n'est pas autorisé
% modern n'est pas chargé
% on reste ici en Pagella
\switchtobodyfont [17.5pt]
Coucou
```

Coucou Coucou

`\setupbodyfont` génère une erreur de compilation si une taille de police non autorisée est demandée ; mais n'en génère pas si une police inexistante est demandée, auquel cas la police par défaut (*Latin Modern Roman*) sera activée. `\switchtobodyfont` agit de la même manière en ce qui concerne la police, et en termes de taille, comme je l'ai déjà dit, essaie d'y parvenir en mettant la police à l'échelle. Cependant, il existe des polices qui ne peuvent pas être mises à l'échelle dans certaines tailles, auquel cas la police par défaut sera à nouveau activée.

6.4.2 Changement rapide de style, d'alternative et de taille

A. Changement de style et de style alternatif

Outre `\switchtobodyfont`, ConTeXt fournit un ensemble de commandes qui nous permettent de changer rapidement le style, le style alternatif ou la taille. En ce qui concerne ces commandes, le wiki ConTeXt nous avertit que parfois, lorsqu'elles apparaissent au début d'un paragraphe, elles peuvent produire des effets secondaires indésirables, et recommande donc que dans ce cas, la commande en question soit précédée de la commande `\dontleavehmode`.

Style	Commandes qui l'active
Romain	<code>\rm, \roman, \serif, \regular</code>
Sans Serif	<code>\ss, \sans, \support, \sansserif</code>
Monospace	<code>\tt, \mono, \teletype,</code>
Handwritten	<code>\hw, \handwritten,</code>
Calligraphique	<code>\cg, \calligraphic</code>

Tableau F.3 Commandes pour changer de styles

Table F.3 contains the commands that allow us to change style, without altering any other aspect ; and table F.4 contains the commands that allow us to exclusively alter the alternative.

Style alternative	Commandes qui l'active
Normal	<code>\tf, \normal</code>
Italique	<code>\it, \italic</code>
Gras	<code>\bf, \bold</code>
Gras-italique	<code>\bi, \bolditalic, \italicbold</code>
Oblique	<code>\sl, \slanted</code>
Gras-oblique	<code>\bs, \boldslanted, \slantedbold</code>
PETITES MAJUSCULES	<code>\sc, \smallcaps</code>
Medieval	<code>\os, \mediaeval</code>

Tableau F.4 Commandes pour changer de style alternatif

Toutes ces commandes conservent leur efficacité jusqu'à ce qu'un autre style ou une autre alternative soit explicitement activé(e), ou jusqu'à ce que le *groupe* dans lequel la commande est déclarée se termine. Par conséquent, lorsque nous voulons que la commande n'affecte qu'une partie du texte, nous devons entourer cette partie d'un groupe, comme dans l'exemple suivant, où chaque fois que le mot *pensée* apparaît alors qu'il s'agit d'un nom et non d'un verbe, il est en italique, ce qui crée un groupe pour lui.

```
J'ai pensé à une {\it pensée}, mais la {\it pensée} que j'ai pensé  
n'était pas la {\it pensée} que je pensais avoir pensé. Si la {\it  
pensée} que je pensais avoir pensé avait été la {\it pensée} que je  
pensais je n'aurais pas pensé autant !
```

J'ai pensé à une *pensée*, mais la *pensée* que j'ai pensé n'était pas la *pensée* que je pensais avoir pensé. Si la *pensée* que je pensais avoir pensé avait été la *pensée* que je pensais je n'aurais pas pensé autant !

B. Commandes pour changer d'alternative et de taille en même temps

Les commandes qui modifient le style alternatif dans leur version à deux lettres (`\tf`, `\it`, `\bf`, etc.) acceptent aussi une gamme de *suffixes* qui affectent la taille de la police.

Les suffixes a, b, c et d augmentent la taille initiale de police en la multipliant respectivement par $1.200^1 = 1.200$, $1.200^2 = 1.440$, $1.200^3 = 1.728$, et $1.200^4 = 2.074$). Les suffixes x et xx réduisent la taille des caractères, en la multipliant respectivement par 0.8 et 0.6. Voici une illustration :

```
\setupbodyfont[modern,12pt]%
```

```
{\tfxx test}, {\tfx test}, {\tf test}, {\tfa test}, {\tfb test}, {\tfc  
test}, {\tfcd test}
```

```
{\tfxx test}, {\itx test}, {\bf test}, {\bia test}, {\slb test}, {\scc  
test}, {\ttd test}
```

test, test, test, test, test, **test**, **test**
test, *test*, **test**, *test*, TEST, **test**

Les suffixes « x » et « xx » appliqués à `\tf` autorisent de raccourcir la commande, de sorte que `\tfx` peut s'écrire `\tx` et `\tfxx \txx`.

La disponibilité de ces différents suffixes dépend de l'implémentation réelle de la police. Selon le manuel de référence ConTeXt 2013 (destiné principalement à Mark II), le seul suffixe dont le fonctionnement est garanti est « x », et les autres peuvent être implémentés ou non ; ou ils peuvent l'être seulement pour certaines alternatives.

En tout cas, pour éviter les doutes, on peut utiliser `\showbodyfont` dont j'ai parlé précédemment (dans [section](#)). Cette commande affiche un tableau qui nous permet non seulement d'apprécier l'apparence de la police, mais aussi de visualiser la police dans chacun de ses styles et alternatives, ainsi que les suffixes de redimensionnement disponibles.

Examinons à nouveau le tableau montrant `\showbodyfont` pour la police *Latin Modern* :

```
\definebodyfontenvironment[14pt]
\showbodyfont[modern,14pt]
```

[modern] [modern, 14pt] \mr : Ag														
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfcd	
\rm	Ag	Ag	Ag	Ag	Ag	Ag	Ag							
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag							
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag							

Si l'on regarde attentivement le tableau, on constate que la première colonne contient les styles de police (`\rm`, `\ss` et `\tt`). La première ligne contient, à gauche, les styles alternatifs (`\tf`, `\sc`, `\sl`, `\it`, `\bf`, `\bs` et `\bi`), tandis que le côté droit de la première ligne contient les autres suffixes disponibles, mais uniquement avec le style alternatif régulier, ou normal.

Il est important de noter qu'un changement de taille de police effectué par l'un de ces suffixes ne modifiera que la taille de la police au sens strict, laissant intactes les autres valeurs normalement associées à la taille de la police, comme l'interligne.

C. Personnalisation des facteurs d'échelle et des suffixes

Pour personnaliser le facteur d'échelle, nous pouvons utiliser `\definebodyfontenvironment` (déjà vu précédemment pour déclarer la taille de la police) dont le format peut être :

```
\definebodyfontenvironment[particular size][scaled]
\definebodyfontenvironment[default][scaled]
```

Dans la première version, nous redéfinissons la mise à l'échelle pour une taille particulière de la police principale définie par `\setupbodyfont` ou par `\switchtobodyfont`. Par exemple :

```
\definebodyfontenvironment [10pt] [a=12pt,b=14pt,c=2, d=3]
```

ferait en sorte que, lorsque la police principale est de 10 points, le suffixe « a » la change en 12 points, le suffixe « b » en 14 points, le suffixe « c » multiplie la police d'origine par 2.0 et le suffixe « d » par 3.0. Notez que pour a et b, une dimension fixe a été indiquée, mais que pour c et d, un facteur de multiplication de la taille d'origine a été indiqué.

Mais si le premier argument de `\definebodyfontenvironment` est égal à « `default` », alors nous redéfinirons la valeur de mise à l'échelle pour toutes les tailles de police possibles, et comme valeur de mise à l'échelle, nous ne pouvons entrer qu'un nombre multiplicateur. Ainsi, si, par exemple, nous écrivons :

```
\definebodyfontenvironment [default] [a=1.3,b=1.6,c=2.5,d=4]
```

nous indiquons que, quelle que soit la taille de la police principale, le suffixe a doit être multiplié par 1.3, le b par 1.6, le c par 2.0 et le d par 4.0.

Outre les suffixes `xx`, `x`, `a`, `b`, `c` et `d`, la commande `\definebodyfontenvironment` permet d'attribuer une valeur d'échelle aux mots clés « `big` », « `small` », « `script` » et « `scriptscript` ». Ces valeurs sont attribuées à toutes les tailles associées à ces mots clés dans `\setupbodyfont` et `\switchbodyfont`. Elles sont également appliquées dans les commandes suivantes, dont l'utilité peut être déduite (je pense) de leur nom :

- `\smallbold`
- `\smallslanted`
- `\smallboldslanted`
- `\smallslantedbold`
- `\smallbolditalic`
- `\smallitalicbold`
- `\smallbodyfont`
- `\bigbodyfont`

Si nous voulons voir les tailles par défaut d'une police particulière, nous pouvons utiliser `\showbodyfontenvironment [Font]`. Cette commande, appliquée à la police `modern`, par exemple, donne le résultat suivant :

```
\definebodyfontenvironment[12pt]
\showbodyfontenvironment[modern,12pt]
```

¹⁴ Nous rappelons que, sauf dans le cas des symboles de contrôle, les noms des commandes ConTeXt peuvent uniquement être

[modern] [modern, 12pt]							
text	script	scriptscript	x	xx	small	big	cointerlinespace
10pt	7pt	5pt	8pt	6pt	8pt	12pt	
11pt	8pt	6pt	9pt	7pt	9pt	12pt	
12pt	9pt	7pt	10pt	8pt	10pt	14.4pt	
14.4pt	11pt	9pt	12pt	10pt	12pt	17.3pt	
17.3pt	12pt	10pt	14.4pt	12pt	14.4pt	20.7pt	
20.7pt	14.4pt	12pt	17.3pt	14.4pt	17.3pt	20.7pt	
4pt	4pt	4pt	4pt	4pt	4pt	6pt	
5pt	5pt	5pt	5pt	5pt	5pt	7pt	
6pt	5pt	5pt	5pt	5pt	5pt	8pt	
7pt	6pt	5pt	6pt	5pt	5pt	9pt	
8pt	6pt	5pt	6pt	5pt	6pt	10pt	
9pt	7pt	5pt	7pt	5pt	7pt	11pt	

6.4.3 Définition de commandes et de mots clés pour les tailles, les styles et les styles alternatifs de polices

Les commandes prédéfinies pour modifier la taille, les styles et les variantes des polices sont suffisantes. De plus, ConTeXt nous permet :

1. d'ajouter notre propre commande de changement de style, de taille ou de style alternatif de police.
2. d'ajouter des synonymes aux noms de styles ou de styles alternatifs reconnus par `\switchtobodyfont`.

ConTeXt fournit les commandes suivantes pour ce faire :

- `\definebodyfontswitch` : nous permet de définir une commande pour changer la taille de la police. Par exemple, si nous voulons définir la commande `\eight` (ou la commande `\viii14`) pour définir une police de 8 points, nous devons écrire :

```
\definebodyfontswitch[quatorze][14pt]
\definebodyfontswitch[xxii][22pt]
{coucou} {\quatorze coucou} {\xxii coucou}
```

coucou coucou COUCOU

- `\definefontstyle` : permet de définir un ou plusieurs mots qui peuvent être utilisés dans `\setupbodyfont` ou `\switchtobodyfont` pour définir un style de police particulier ; ainsi, par exemple, si nous voulons appeler la police *sans serif* autrement (par exemple, en français, nous pourrions l'appeler « lineale » ou « sansempattement »), nous pouvons créer des synonymes en écrivant :

```
\setupbodyfont[modern,12pt]%
\definefontstyle[lineale,sansempattement][ss]
coucou
\setupbodyfont[lineale]
coucou
```

coucou coucou

- `\definealternativestyle`: permet d'associer un nom à un style alternatif de police. Ce nom peut fonctionner comme une commande ou être reconnu par l'option `style` des commandes qui nous permettent de configurer le style à appliquer. Ainsi, par exemple, le fragment suivant

```
\setupbodyfont[modern,12pt]
\definealternativestyle[strong][\bf] []
coucou \strong coucou
```

coucou **coucou**

activera la commande `\strong` et le mot clé « `strong` » qui sera reconnu par l'option `style` des commandes qui autorisent cette option. Nous aurions pu dire « `bold` » mais ce mot est déjà utilisé pour ConTeXt, j'ai donc choisi un terme utilisé en HTML, à savoir, « `strong` » comme alternative.

Je ne sais pas ce que fait le troisième argument de `\definealternativestyle`. Il n'est pas optionnel et ne peut donc pas être omis ; mais la seule information que j'ai trouvée à ce sujet se trouve dans le manuel de référence ConTeXt où il est dit que ce troisième argument ne concerne que les titres de chapitre et de section « *où, en dehors de \cap, nous devons respecter la police utilisée ici* ». (??)

6.5 Autres questions relatives à l'utilisation de styles alternatifs

Parmi les différents styles alternatifs d'une police de caractères, il en existe deux dont l'utilisation nécessite certaines précisions :

6.5.1 Italique, oblique et mise en valeur

L'italique et l'oblique sont utilisées principalement pour mettre en évidence un fragment de texte afin d'attirer l'attention sur celui-ci. En d'autres termes, pour le mettre en valeur.

Nous pouvons, bien sûr, mettre en valeur un texte en activant explicitement l'italique ou l'oblique. Mais ConTeXt offre une commande alternative qui est beaucoup plus utile et intéressante et qui est destinée spécifiquement à mettre en valeur un fragment de texte. Il s'agit de la commande `\em` du mot *emphasis*. Contrairement à `\it` et `\sl`, qui sont des commandes purement typographiques, `\em` est une commande *conceptuelle* ; elle fonctionne différemment et est plus polyvalente, au point que la documentation ConTeXt recommande d'utiliser `\em` de préférence à `\it` ou `\sl`. Lorsque nous utilisons ces deux dernières commandes, nous indiquons à ConTeXt quelle alternative de police nous voulons utiliser ; mais lorsque nous utilisons `\em`, nous lui indiquons l'effet que nous voulons produire, en laissant à ConTeXt le soin de décider comment le faire. Normalement, pour obtenir l'effet de mise en valeur de quelque chose, nous activerions l'italique ou l'oblique, mais cela dépend du contexte. Ainsi, si nous utilisons `\em` dans un texte qui est déjà en italique – ou oblique – la commande le mettra en évidence de la manière opposée – en texte droit normal dans ce cas.

D'où l'exemple suivant :

```
{\it L'une des plus belles  
orchidées du monde est la  
{\em Thelymitra variegata}  
ou Reine de Saba du Sud.}
```

L'une des plus belles orchidées du monde est la Thelymitra variegata ou Reine de Saba du Sud.

Notez que le premier `\em` active l'italique (en fait, l'oblique, mais voir ci-dessous) et que le second `\em` le désactive et place les mots « Thelymitra variegata » dans un style droit normal.

Un autre avantage de `\em` est qu'il ne s'agit pas d'un style alternatif, donc il ne désactive pas l'alternative que nous avions auparavant et donc, par exemple, dans un texte qui est en gras, avec `\em` nous obtiendrons du gras oblique sans avoir besoin de faire explicitement appel à `\bs`. De même, si la commande `\bf` apparaît dans un texte qui est déjà mis en valeur, celle-ci ne cessera pas.

```
{\bf L'une des plus belles  
orchidées du monde est la  
{\em Thelymitra variegata}  
ou Reine de Saba du Sud.}
```

L'une des plus belles orchidées du monde est la Thelymitra variegata ou Reine de Saba du Sud.

Par défaut, `\em` active le gras oblique plutôt que l’italique, mais nous pouvons modifier cela avec `\setupbodyfontenvironment [default] [em=italic]`.

6.5.2 Petites majuscules et fausses petites majuscules

Les petites majuscules sont une ressource typographique qui est souvent bien meilleure que l’utilisation des lettres majuscules (capitales). Les petites majuscules nous donnent la forme de la lettre majuscule mais conservent la même hauteur que les lettres minuscules sur la ligne. C’est pourquoi les petites majuscules sont un style alternatif des minuscules. Les petites majuscules remplacent les majuscules dans certains contextes, et sont particulièrement utiles pour écrire les chiffres romains ou les titres de chapitres. Dans les textes universitaires, il est également d’usage d’utiliser les petites majuscules pour écrire le nom des auteurs cités.

Le problème est que toutes les polices de caractères n’intègrent pas les petites majuscules, et celles qui le font ne le font pas toujours pour l’ensemble de leurs styles de police. De plus, les petites majuscules étant une alternative à l’italique, au gras ou à l’oblique, selon les règles générales que nous avons énoncées dans ce chapitre, toutes ces caractéristiques typographiques ne peuvent être utilisées simultanément.

Ces problèmes peuvent être résolus par l’utilisation de *fausses petites capitales* que ConTeXt nous permet de créer avec les commandes `\cap` et `\Cap`; à cet égard, voir section 10.2.1.

6.6 Utilisation et configuration des couleurs

ConTeXt fournit des commandes pour changer la couleur d'un document entier, de certains de ses éléments ou de certaines parties du texte. Il fournit également des commandes permettant de mettre en mémoire des centaines de couleurs prédéfinies (section 6.6.4) et de voir quels sont leurs composants.

6.6.1 Utiliser des couleurs pour des éléments textuels

La plupart des commandes configurables de ConTeXt comportent une option appelée « `color` » qui nous permet d'indiquer la couleur dans laquelle le texte affecté par cette commande doit être écrit. Ainsi, par exemple, pour indiquer que les titres de chapitre sont écrits en bleu, il suffit d'écrire :

```
\setuphead  
  [chapter]  
  [color=blue]
```

Cette méthode permet de colorer les titres, les en-têtes, les notes de bas de page, les notes de marge, les barres et les lignes, les tableaux, les titres de tableaux ou d'images, etc. L'avantage d'utiliser cette méthode est que le résultat final sera cohérent (tous les textes qui remplissent la même fonction seront écrits avec la même couleur) et plus facile à modifier globalement.

On peut également colorer directement une portion ou un fragment de texte avec la commande `\color`, bien que, pour éviter une utilisation trop variée des couleurs, peu agréable du point de vue typographique, ou une utilisation incohérente, il est généralement recommandé d'éviter la coloration directe et d'utiliser ce que l'on pourrait appeler la *coloration sémantique*, c'est-à-dire qu'au lieu d'écrire par exemple :

```
\color[red]{Very important text}
```

Very important text

nous définissons une commande spécifique avec `\definehighlight` auquel nous associons une couleur. Pour exemple :

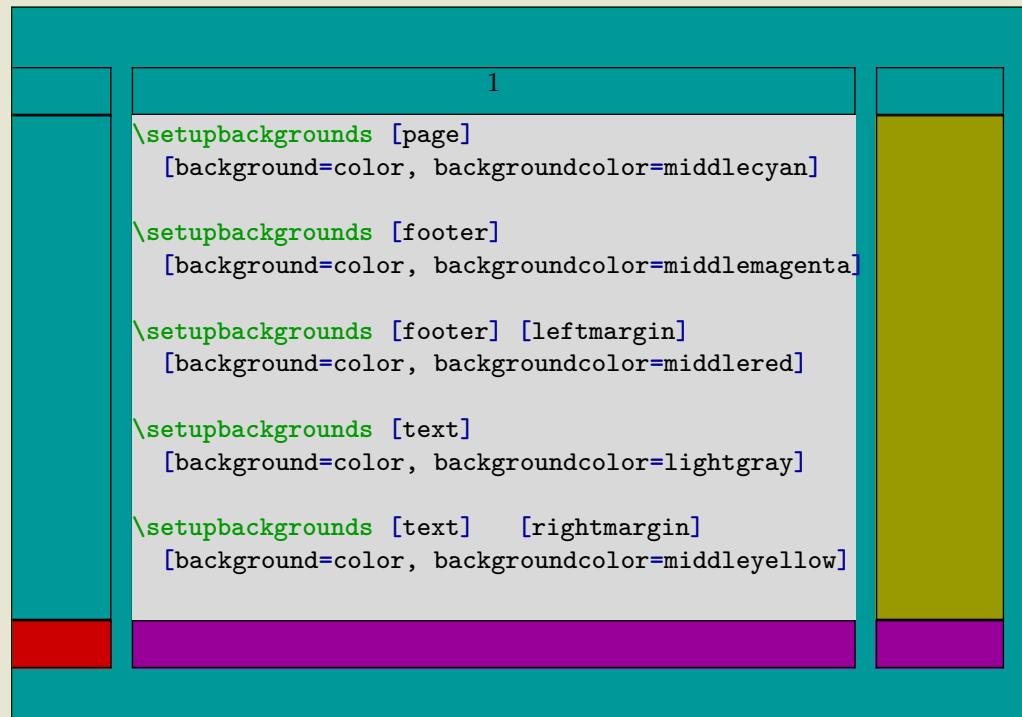
```
\definehighlight[important][color=red]  
\important{Very important text}
```

Very important text

Ainsi, pour modifier de façon cohérente l'ensemble des textes indiqués comme « important » dans le code source, il suffira de modifier la déclaration de `\definehighlight`.

6.6.2 Utiliser des couleurs en arrière-plan et pour le texte en général

Si nous voulons changer la couleur de l'ensemble du document, selon que nous voulons modifier la couleur de l'arrière-plan ou celle du premier plan (texte), nous utiliserons `\setupbackgrounds` ou `\setupcolors`. Ainsi, par exemple



Cette commande définit la couleur de fond des pages comme étant le cyan, et vous voyez comment il est possible d'affecter une couleur à chaque zone de la page vue à la figure [section 5.1 page 109](#). Comme valeur pour « `backgroundcolor` », nous pouvons utiliser le nom de l'une des couleurs prédéfinies ([section 6.6.4](#)).

Pour modifier globalement la couleur d'avant-plan dans tout le document (à partir de l'endroit où la commande est insérée), utilisez `\setupcolors`, où l'option « `textcolor` » contrôle la couleur du texte. Par exemple :

```
\setupcolors [textcolor=middlecyan]
Texte coloré
```

Texte coloré

6.6.3 Utiliser des couleurs pour des portions de texte

Comme nous l'avons vu précédemment La commande générale pour colorier de petites portions de texte est la suivante :

```
\color [ColourName] {Text to colour}
```

Pour les grandes portions de texte, il est préférable d'utiliser l'environnement « `color` » avec `\startcolor` et `\stopcolor`.

```
\startcolor[ColourName] ... \stopcolor
```

Ces deux commandes utilisent des couleurs prédéfinies que l'on désigne par leur nom ([section 6.6.4](#)). Si nous voulons définir la couleur à la volée, nous pouvons utiliser la commande `\colored`. Par exemple :

```
Trois chats  
\colored[r=0.1, g=0.8, b=0.8]{colorés}.
```

Trois chats colorés.

¹⁵ Cette liste se trouve dans le manuel de référence et le wiki ConTeXt mais je suis presque sûr qu'il s'agit d'une liste incomplète puisque dans ce document, par exemple, sans avoir chargé de couleur supplémentaire, nous utilisons « orange » – qui n'est pas dans le [tableau 6.5](#) – pour les titres de section.

6.6.4 Couleurs prédéfinies

ConTeXt charge les couleurs prédéfinies les plus courantes listées dans le [tableau 6.5](#).¹⁵

Nom	Tonalité claire	Tonalité moyenne	Tonalité foncée
black			
white			
gray	lightgray	middlegray	darkgray
red	lightred	middlered	darkred
green	lightgreen	middlegreen	darkgreen
blue	lightblue	middleblue	darkblue
cyan		middlecyan	darkcyan
magenta		middlemagenta	darmagenta
yellow		middleyellow	darkyellow

Tableau 6.5 ConTeXt's predefined colours

Il existe d'autres collections de couleurs qui ne sont pas chargées par défaut mais qui peuvent être chargées avec la commande `\usecolors[CollectionName]` où « `CollectionName` » peut être :

- « `crayola` », 235 couleurs imitant les nuances des marqueurs.
- « `dem` », 91 couleurs.
- « `ema` », 540 couleurs basées sur celles utilisées par Emacs.
- « `rainbow` », 91 couleurs à utiliser dans les formules de mathématiques.
- « `ral` », 213 couleurs provenant du *Deutsches Institut für Gütesicherung und Kennzeichnung* (Institut allemand pour l'assurance qualité et l'étiquetage).
- « `rgb` », 223 couleurs.
- « `solarized` », 16 couleurs basées sur le schéma solarized.
- « `svg` », 147 couleurs.
- « `x11` », 450 couleurs standard pour X11.
- « `xwi` », 124 couleurs.

Les fichiers de définition des couleurs sont inclus dans le répertoire « `context/base/mkiv` » de la distribution et son nom répond au schéma « `colo-imp-NOMBRE.mkiv` ». Les informations que je viens de fournir sur les différentes collections de couleurs prédéfinies sont basées sur ma distribution particulière. Les collections spécifiques, ou le nombre de couleurs définies dans celles-ci, pourraient changer dans les versions futures.

Pour voir quelles couleurs contiennent chacune de ces collections, nous pouvons utiliser la commande `\showcolor[CollectionName]` décrite dans ce qui suit [section 6.6.5](#).

Pour utiliser certaines de ces couleurs, il faut d'abord les charger en mémoire avec la commande (`\usecolors[CollectionName]`), puis indiquer le nom de la couleur aux commandes `\color` ou `\startcolor`. Par exemple, la séquence suivant :

```
\usecolors[xwi]
\color[darkgoldenrod]{Tweedledum and Tweedledee}
```

Tweedledum and Tweedledee

6.6.5 Visualier les couleurs disponibles

La commande `\showcolor` affiche une liste de couleurs dans laquelle vous pouvez voir l'apparence de la couleur, son apparence lorsque la couleur est passée en échelle de gris (impression noir et blanc par exemple), les composantes rouge, verte et bleue de la couleur, ainsi que le nom par lequel ConTeXt la connaît. Utilisée sans argument, `\showcolor` affichera les couleurs utilisées dans le document actuel. Mais comme argument, nous pouvons indiquer l'une des collections prédefinies de couleurs qui ont été discutées dans [section 6.6.4](#), et ainsi, par exemple, `\showcolor[solarized]` nous montrera les 16 couleurs de cette collection :

```
\usecolors[solarized]
\showcolor[solarized]
```



0.561	0.514	0.580	0.588	base0
0.460	0.396	0.482	0.514	base00
0.409	0.345	0.431	0.459	base01
0.162	0.027	0.212	0.259	base02
0.123	0.000	0.169	0.212	base03
0.615	0.576	0.631	0.631	base1
0.909	0.933	0.910	0.835	base2
0.965	0.992	0.965	0.890	base3
0.457	0.149	0.545	0.824	blue
0.487	0.165	0.631	0.596	cyan
0.510	0.522	0.600	0.000	green
0.429	0.827	0.212	0.510	magenta
0.422	0.796	0.294	0.086	orange
0.395	0.863	0.196	0.184	red
0.473	0.424	0.443	0.769	violet
0.530	0.710	0.537	0.000	yellow

Si nous voulons voir les composantes `rgb` d'une couleur particulière, nous pouvons utiliser `\showcolorcomponents[ColourName]`. Ceci est utile si nous essayons de définir une couleur spécifique, pour voir la composition d'une couleur qui lui est proche. Par exemple, `\showcolorcomponents[darkgoldenrod]` nous montrera :

```
\usecolors[xwi]
\showcolorcomponents[darkgoldenrod]
```

```
color           name          transparency  specification
white black    darkgoldenrod   r=0.720,g=0.530,b=0.040
```

6.6.6 Definir ses propres couleurs

`\definecolor` nous permet soit de cloner une couleur existante, soit de définir une nouvelle couleur. Cloner une couleur existante est aussi simple que de lui donner un autre nom. Pour ce faire, vous devez écrire :

```
\definecolor[NouvelleCouleur][AncienneCouleur]
```

Ainsi, "NouvelleCouleur" sera exactement de la même couleur que "AncienneCouleur".

Mais la principale utilisation de `\definecolor` est la création de nouvelles couleurs. Pour ce faire, la commande doit être utilisée de la manière suivante :

```
\definecolor[ColourName][Définition]
```

où *Définition* peut se faire en appliquant jusqu'à six schémas de génération de couleurs différents :

1. **Couleurs RVB** : La définition des couleurs RVB est l'une des plus répandues ; elle repose sur l'idée qu'il est possible de représenter une couleur en mélangeant, par addition, les trois couleurs primaires : rouge (« r » pour *rouge*), vert (« g » pour *vert*) et bleu (« b » pour *bleu*). Chacun de ces composants est indiqué par un nombre décimal compris entre 0 et 1.

```
\definecolor [CouleurA]
[r=0.720, g=0.530, b=0.040]
\color[CouleurA]{Texte Couleur A.}
```

Texte Couleur A.

2. **Couleurs hexadécimales** : Cette façon de représenter les couleurs est également basée sur le schéma RVB, mais les composantes rouge, verte et bleue sont indiquées sous la forme de trois nombres hexadécimaux, le premier représentant la valeur du rouge, le deuxième la valeur du vert et le troisième la valeur du bleu. Par exemple :

```
\definecolor [CouleurB]
[x=B8860B]
\color[CouleurB]{Texte Couleur B.}
```

Texte Couleur B.

3. **Couleurs CMYK** : Ce modèle de génération des couleurs est ce qu'on appelle un « modèle soustractif » et repose sur le mélange de pigments des couleurs suivantes : cyan (« c »), magenta (« m »), jaune (« y », de *yellow*) et noir (« k », de *key* (key au sens valeur)). Chacun de ces composants est indiqué par un nombre décimal compris entre 0 et 1 :

```
\definecolor [CouleurC]
[c=0.00, m=0.20, y=0.68, k=0.28]
\color [CouleurC]{Texte Couleur C.}
```

Texte Couleur C.

4. **Couleurs HSL/HSV** : Ce modèle de couleur est basé sur la mesure de la teinte (« h », de *hue*), de la saturation (« s ») et de la luminescence (« l » ou parfois « v », de *value*). La teinte correspond à un nombre compris entre 0 et 360 ; la saturation et la luminescence doivent être un nombre décimal compris entre 0 et 1. Par exemple :

```
\definecolor [CouleurD] [h=43.00, s=0.89,
v=0.38]
\color [CouleurD]{Texte Couleur D.}
```

Texte Couleur D.

5. **Couleurs HWB** : Le modèle HWB est une norme suggérée pour CSS4 qui mesure la teinte (« h », de *hue*), et le niveau de blanc (« w », de *whiteness*) et de noir (« b », de *blackness*). La teinte correspond à un nombre compris entre 0 et 360, tandis que la blancheur et la noirceur sont représentées par un nombre décimal compris entre 0 et 1.

```
\definecolor [CouleurE] [h=43.00, w=0.04,
b=0.28]
\color [CouleurE]{Texte Couleur E.}
```

Texte Couleur E.

6. **Couleur échelle de gris** : basé sur un composant appelé (« s », de *scale*) qui mesure la quantité de gris. Il doit s'agir d'un nombre compris entre 0 et 1. Par exemple :

```
\definecolor [CouleurF] [s=0.65] %
\color [CouleurF]{Texte Couleur F.}
```

Texte Couleur F.

Il est également possible de définir une nouvelle couleur à partir d'une autre couleur. Par exemple, la couleur dans laquelle les titres sont écrits dans cette introduction est définie comme suit

```
\definecolor [CouleurG] [0.8(orange)] %
\definecolor [CouleurH] [0.6(orange)] %
\definecolor [CouleurI] [0.4(orange)] %
\definecolor [CouleurJ] [0.2(orange)] %
\color [CouleurG]{Texte Couleur G.} \\
\color [CouleurH]{Texte Couleur H.} \\
\color [CouleurI]{Texte Couleur I.} \\
\color [CouleurJ]{Texte Couleur J.}
```

Texte Couleur G.

Texte Couleur H.

Texte Couleur I.

Texte Couleur J.

6.7 Bonus 1 - Utilisation des polices du système d'exploitation

6.7.1 Emplacement des polices sur votre ordinateur

La première étape consiste à déclarer les emplacements de stockage des polices que vous voulez que ConTeXt prenne en compte.

Dans tous les cas, ConTeXt utilisera les polices correctement stockées dans son arborescence (par exemple, toutes les polices que vous auriez téléchargées à partir de [Fonts Squirrel](#) ou encore [Google Fonts](#)).

Les utilisateurs de TeX créent un nouveau dossier pour chaque nouvelle police dans « `tex/texmf-fonts/fonts/` », en suivant la [structure de répertoire de TeX](#). Cela aide les algorithmes à gérer l'incroyable variété de variables et de paramètres des polices. Les personnes qui manipulent beaucoup de polices peuvent être plus structurées en décomposant encore plus finement le chemin par exemple en utilisant « `tex/texmf-fonts/fonts/truetype/vendor/fontfamily` ».

Mais il est très probable que vous souhaitiez également utiliser les polices déjà disponibles sur votre système d'exploitation :

1. Spécifiez où ConTeXt doit chercher les polices, en définissant la variable d'environnement « `OSFONDIR` ».

– WINDOWS :

```
set OSFONDIR=c:/windows/fonts/
```

– MAC :

```
export OSFONDIR=/Library/Fonts/:/System/Library/Fonts:$HOME/Library/Fonts
```

– GNU/LINUX :

```
export OSFONDIR=$HOME/.fonts:/usr/share/fonts
```

– Ajoutez-le à votre `.bashrc` ou à l'équivalent shell pour rendre la déclaration permanente.

2. Lancez ConTeXt pour indexer les fichiers et les polices.

```
mtxrun --generate  
mtxrun --script font --reload
```

3. Vérifiez en cherchant la police spécifique que vous voulez utiliser ensuite. Un exemple courant

```
mtxrun --script font --list --file -pattern=*helvetica*.
```

Maintenant, apprenons à les utiliser.

6.7.2 Utilisation rapide d'une nouvelle police de caractères

Prenons un exemple : nous voulons utiliser la police [Noto Serif](#).

Si elle est déjà installé sur votre ordinateur, et que vous avez déjà mis à jour les bases de données ConTeXt comme indiqué précédemment, allez directement au point 2.

Sinon, vous devez d'abord la télécharger et la stocker. Le site de Google fournit un fichier zip avec les 4 variations alternatives (Regular 400, Regular 400 italic, Bold 700, Bold 700 italic).

1. Stockez-les dans un dossier dédié indexé par ConTeXt (voir ci-dessus).
 - par exemple, créez un répertoire « Noto-serif » dans la distribution ConTeXt « `tex/texmf-fonts/fonts/` » (ou bien, sous LINUX, dans « `/.fonts` »).
 - dézippez et stockez les fichiers .ttf dans « `tex/texmf-fonts/fonts/Noto-serif/` ».
 - Régénérer les bases de données ConTeXt

```
mtxrun --generate  
mtxrun --script font --reload
```

2. Maintenant vous pouvez vérifier le nom de la police utilisé pour identifier les polices, en lançant le script mtxrun :

```
mtxrun --script fonts --list --all --pattern=*notoserif  
identifier      familyname   fontname      filename      subfont  
instances  
  
notoserif       notoserif    notoserif      NotoSerif-Regular.ttf  
notoserifbold   notoserif    notoserifbold  NotoSerif-Bold.ttf  
notoserifbolditalic  notoserif    notoserifbolditalic  NotoSerif-BoldItalic.ttf  
notoserifitalic  notoserif    notoserifitalic  NotoSerif-Italic.ttf
```

3. Vous pouvez maintenant utiliser la police n'importe où dans vos fichiers sources avec la commande `\definedfont [name :lefontname*default]` (il est bon d'ajouter « `*default` » pour bénéficier des fonctionnalités par défaut, comme par exemple le crénage (kerning)).

```
\definedfont [name:notoserifbolditalic*default at 12 pt]%
Le renard brun et rapide saute par-dessus le chien paresseux.
```

Le renard brun et rapide saute par-dessus le chien paresseux.

6.7.3 Utilisation de divers styles alternatifs de police

Il n'est pas agréable de devoir écrire `\definedfont[name : mapolice-graissesstyle*default at xxpt]` chaque fois que vous voulez utiliser une police particulière. C'est pourquoi il est utile de définir un *typescript*. C'est juste 3 étapes, et moins de 5 minutes. Ensuite, vous pourrez facilement passer d'un style ou style alternative à l'autre avec les commandes vues précédemment, et toute la typographie de votre document utilisera un ensemble cohérent de polices. De nombreuses polices de caractères sont prêtes à être utilisées avec les polices libres et commerciales habituelles, et évidemment avec celle de « ConTeXt Standalone ».

1. Définissez un nouveau *typescript* dans votre fichier d'entrée, avec `\starttypescript`.
 - Définissez les liens entre les noms de fichiers et les noms lisibles par le public avec `\definefontsynonym`.
 - Dans cet exemple, le *typescript* s'appelle « mynotoserif ».
 - Rappel : vous trouvez les noms de fichiers pour les polices Noto Serif avec « `mtxrun --script fonts --list --all --pattern=*notoserif` »

```
\starttypescript [mynotoserif]
% \definefontsynonym[Human readable]           [file:filename without extension]
\definefontsynonym[NotoSerif-Regular]         [file:NotoSerif-Regular]
\definefontsynonym[NotoSerif-Italic]          [file:NotoSerif-Italic]
\definefontsynonym[NotoSerif-Bold]             [file:NotoSerif-Bold]
\definefontsynonym[NotoSerif-BoldItalic]       [file:NotoSerif-BoldItalic]
\stoptypescript
```

C'est ici que vous pouvez identifier notamment les alternatives « thin », « extralight », « light », « medium », « semi-bold », « extrabold », « black », « ultra-black », « condensed », « extracondensed ».

2. L'étape ennuyeuse, définir les liens entre les noms de base ConTeXt et les noms compréhensible par l'utilisateur. Une bonne habitude à prendre consiste à bien définir une solution de repli (au cas où la police indiquée ne serait pas accessible à ConTeXt).

```
\starttypescript [mynotoserif]
\setup[font:fallback:serif]           % security: if not found=> back to defaults
% \definefontsynonym[ConTeXt basics name] [Human readable]      [features=default]
\definefontsynonym[Serif]              [NotoSerif-Regular]     [features=default]
\definefontsynonym[SerifItalic]        [NotoSerif-Italic]      [features=default]
\definefontsynonym[SerifBold]          [NotoSerif-Bold]        [features=default]
\definefontsynonym[SerifBoldItalic]    [NotoSerif-BoldItalic]  [features=default]
\stoptypescript
```

3. Définir le pack des 4 styles alternatifs comme le caractère « romain » ou « serif » du *typescript* « mynotoserif ».

```
\starttypescript [mynotoserif]
  \definetypeface [mynotoserif] [rm] [serif] [mynotoserif] [default]
\stoptypescript
```

4. au final, nous disposons maintenant d'un *typescript* utilisable :

```
\starttypescript [mynotoserif]
  \definefontsynonym[NotoSerif-Regular] [file:NotoSerif-Regular]
  \definefontsynonym[NotoSerif-Italic] [file:NotoSerif-Italic]
  \definefontsynonym[NotoSerif-Bold] [file:NotoSerif-Bold]
  \definefontsynonym[NotoSerif-BoldItalic] [file:NotoSerif-BoldItalic]
\stoptypescript

\starttypescript [mynotoserif]
  \setup[font:fallback:serif]
  \definefontsynonym[Serif] [NotoSerif-Regular] [features=default]
  \definefontsynonym[SerifItalic] [NotoSerif-Italic] [features=default]
  \definefontsynonym[SerifBold] [NotoSerif-Bold] [features=default]
  \definefontsynonym[SerifBoldItalic] [NotoSerif-BoldItalic] [features=default]
\stoptypescript

\starttypescript [mynotoserif]
  \definetypeface [mynotoserif] [rm] [serif] [mynotoserif] [default]
\stoptypescript

\setupbodyfont[mynotoserif]
\setupbodyfont[12pt]
{ The quick brown fox jumps over the lazy dog} \\
{\it The quick brown fox jumps over the lazy dog} \\
{\bf The quick brown fox jumps over the lazy dog} \\
{\bi The quick brown fox jumps over the lazy dog}
```

The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog

5. à vous de poursuivre pour construire un ensemble complet présentant les styles « Sans Serif », « Monospace », « Handwritten », « Calligraphique ».

6.7.4 Installation d'un *typescript* pour l'utiliser partout

Vous voudrez probablement utiliser vos nouvelles définitions de caractères personnalisées dans différents documents, et vous devrez donc les installer dans la distribution. Ici, nous rappelons la définition :

- Enregistrez votre fichier sous le nom de « type-imp-(un nom quelconque).tex », par exemple ici « type-imp-mynotoserif.tex ».
- Copiez les fichiers typescript dans « tex/texmf-fonts/tex/context/user/ ».

- Exécutez « context --generate » pour mettre à jour la base de données des fichiers ConTeXt.
- C'est fait ! Maintenant, deux lignes au début de n'importe quelle entrée déclareront qu'il faut composer avec les nouvelles polices :

```
\usetypescriptfile[mynotoserif] % this is the 'some-name-you-like' part of the saved filename  
\setupbodyfont[mynotoserif]      % this is the first argument to \definetypeface
```

6.7.5 Quelques dernières fonctionnalités avec les polices

Certaines polices proposent des « fonctionnalités » très spécifiques et chères aux amateurs typographes, voyez par exemple l'utilisation suivante avec la police Garamond Premier, qu'il faudrait développer petit à petit :

```

\definefontfeature
[mesfeaturesA]
[mode=node,
language=dflt,
protrusion=quality, % for protrusion (dans les marges)
expansion=quality, % for expansions (expansion des lettres)
script=latn,
kern=no, % for kerning
liga=no, % ligatures communes
dlig=no, % ligatures spécifiques (exemple st)
calt=no, % alternatives contextuelles
lnum=yes, % lining numbers
onum=no, % old style numbers,
ccmp=no, % petites majuscules
ss04=no, % stylistic swash
]

\definefontfeature
[mesfeaturesB]
[mode=node,
language=dflt,
protrusion=quality, % for kerning
expansion=quality, % for kerning
script=latn,
kern=yes, % for kerning
liga=yes, % ligatures communes
dlig=yes, % ligatures spécifiques (exemple st)
calt=yes, % alternatives contextuelles
lnum=no, % lining numbers
onum=yes, % old style numbers,
ccmp=yes, % petites majuscules
ss04=yes, % stylistic swash
]

\definedfont [name:garamondpremrpro*mesfeaturesA at 24 pt]%
st ffl fi fj fh Qu 0123456789

\definedfont [name:garamondpremrpro*mesfeaturesB at 24 pt]%
st ffl fi fj fh Qu 0123456789

```

st ffl fi fj fh Qu 0123456789
 st ffl fi fj fh Qu 0123456789

Pour plus de détail, notamment pour activer ou désactiver localement certaines fonctionnalités, n'hésitez pas à consulter le [wiki](#).

Chapitre 7

Structure du document

Table of Contents : [7.1 Les divisions structurelles d'un document](#) ; [7.2 Types et hiérarchie des sections](#) ; [7.3 Syntaxe commune des commandes liées aux sections](#) ; [7.4 Format et configuration des sections et de leurs titres](#) ; [7.4.1 Les commandes \setuphead and \setupheads](#) ; [7.4.2 Parties du titre d'une section](#) ; [7.4.3 Contrôle de la numérotation \(dans les sections numérotées\)](#) ; A Règles de remise à zéro des compteurs ; B Règles de numérotation ; C Séparateurs ; D Raffinements ; [7.4.4 Couleur et style du titre](#) ; [7.4.5 Emplacement du numéro et du texte du titre](#) ; [7.4.6 Commandes ou actions à effectuer avant ou après un titre ou une section](#) ; [7.4.7 Autres fonctionnalités configurables](#) ; [7.4.8 Autres options de \setuphead](#) ; [7.5 Définir de nouvelles commandes de section](#) ; [7.6 La macrostructure du document](#) ;

7.1 Les divisions structurelles d'un document

À l'exception des textes très courts (comme une lettre, par exemple), un document est généralement structuré en blocs ou en groupements de textes qui suivent généralement un ordre hiérarchique. Il n'y a pas de manière standard de nommer ces blocs : dans les romans, par exemple, les divisions structurelles sont généralement appelées « chapitres » bien que certains – les plus longs – aient des blocs plus grands généralement appelés « parties » qui regroupent un certain nombre de chapitres. Les œuvres théâtrales font la distinction entre les « actes » et les « scènes ». Les manuels universitaires sont divisés (parfois) en « parties » et « leçons », « sujets » ou « chapitres » qui, à leur tour, ont souvent des divisions internes également ; le même type de divisions hiérarchiques complexes existe souvent dans d'autres documents universitaires ou techniques (tels que des textes comme le présent texte consacré à l'explication d'un programme ou d'un système informatique. Même les lois sont structurées en « livres », (les plus longs et les plus complexes, comme les Codes), « titres », « chapitres », « sections », « sous-sections ». Les documents scientifiques et techniques peuvent également atteindre jusqu'à six, sept ou même parfois huit niveaux de profondeur.

Ce chapitre se concentre sur l'analyse du mécanisme que propose ConTEXt pour mettre en oeuvre ces divisions structurelles. Je les désignerai par le terme général de « sections ».

Il n'existe pas de terme clair qui nous permette de nous référer de manière générique à tous ces types de divisions structurelles. Le terme « section », pour lequel j'ai opté, se concentre sur la division structurelle plutôt que sur autre chose, bien qu'un inconvénient soit que l'une des divisions structurelles prédéterminées de ConTeXt soit justement appelée une « section ». J'espère que cela ne créera pas de confusion, car je pense qu'il sera assez facile de déterminer à partir du contexte si nous parlons de section en tant que référence générique et globale aux divisions structurelles, ou d'une division spécifique que ConTeXt appelle une section.

Chaque « section » (de manière générique) implique :

- Une *division structurelle du document* raisonnablement grande d'un document qui peut, à son tour, inclure d'autres divisions de niveau inférieur. Dans cette perspective, les *sections* impliquent des blocs de texte avec une relation hiérarchique entre eux. Du point de vue de ses sections, le document dans son ensemble peut être considéré comme un arbre. Le document *en soi* est le tronc, chacun de ses chapitres une branche, qui à son tour peut avoir des rameaux qui peuvent aussi se subdiviser et ainsi de suite.
Il est très important d'avoir une structure claire pour que le document puisse être lu et compris. Cette tâche incombe toutefois à l'auteur, et non au compositeur. Et bien qu'il ne revienne pas à ConTeXt de faire de nous de meilleurs auteurs que nous ne le sommes, la gamme complète de commandes de section qu'il inclut, où la hiérarchie entre elles est très claire, pourrait nous aider à écrire des documents mieux structurés.
- Un *nom de structure* que nous pourrions appeler son « titre » ou « label ». Ce nom de structure est affiché :
 - Toujours (ou presque toujours) à l'endroit du document où commence la division structurelle.
 - Parfois aussi dans la table des matières, dans l'en-tête ou le pied de page des pages occupées par la section en question.

ConTeXt nous permet d'automatiser toutes ces tâches de telle sorte que les caractéristiques de formatage avec lesquelles le titre d'une unité structurelle doit être affiché (où que ce soit et notamment dans la table des matières, ou dans les en-têtes ou les pieds de page) ne doivent être indiquées qu'une seule fois. Pour ce faire, ConTeXt a seulement besoin de savoir où commence et finit chaque unité structurelle, comment elle s'appelle et à quel niveau hiérarchique elle se situe.

7.2 Types et hiérarchie des sections

ConTeXt fait la distinction entre les sections *numérotées* et *non numérotées*. Les premières, comme leur nom l'indique, sont numérotées automatiquement et envoyées à la table des matières, ainsi que, parfois, aux en-têtes et/ou pieds de page.

ConTeXt a des commandes de section prédéfinies et hiérarchisées qui se trouvent dans la [table 7.1](#).

Niveau	Sections numérotées	Sections non numérotées
1	\part	-
2	\chapter	\title
3	\section	\subject
4	\subsection	\subsubject
5	\subsubsection	\subsubsubject
6	\subsubsubsection	\subsubsubsubject
...

Tableau 7.1 Section commands in ConTeXt

En ce qui concerne les sections prédéfinies, les précisions suivantes doivent être apportées :

- Dans le [tableau 7.1](#), les commandes de section sont présentées sous leur forme traditionnelle. Mais nous verrons tout de suite qu'elles peuvent également être utilisées comme des *environnements* (\startchapter ... \stopchapter, par exemple) et que c'est l'approche qui est réellement recommandée.
- Le tableau ne contient que les 6 premiers niveaux de section. Dans mes tests, cependant, j'ai trouvé jusqu'à 12 niveaux : Après \subsubsubsection vient \subsubsubsubsection, et ainsi de suite jusqu'à \subsubsubsubsubsection, ou \subsubsubsubsubsubsection, ou \subsubsubsubsubsubsubsection.

Mais il ne faut pas oublier que les niveaux inférieurs (trop profonds) indiqués ci-dessus ne sont guère susceptibles d'améliorer la compréhension d'un texte ! Tout d'abord, nous risquons d'avoir de grandes sections traitant inévitablement de plusieurs sujets, ce qui rendra difficile pour le lecteur d'en *saisir* le contenu. En outre, si l'on approfondit excessivement les niveaux, le lecteur risque de perdre le sens global du texte, et l'effet produit est celui d'une fragmentation excessive du matériel concerné. Je crois savoir qu'en général, quatre niveaux sont suffisants ; très occasionnellement, il peut être nécessaire d'aller jusqu'à six ou sept niveaux, mais une plus grande profondeur est rarement une bonne idée.

Du point de vue de l'écriture du fichier source, le fait que la création de sous-niveaux supplémentaires signifie l'ajout d'une autre « sub » au niveau précédent peut rendre le fichier source presque illisible : ce n'est pas une blague d'essayer de déterminer le niveau d'une commande nommée « subsubsubsubsubsection » puisque je dois compter toutes les « subs » ! Mon conseil est donc que si nous avons vraiment besoin de tant de niveaux de profondeur, à partir du cinquième niveau (sous-sous-section), nous ferions mieux de définir nos propres commandes de section (voir [section 7.5](#)) en leur donnant des noms plus clairs que les noms prédéfinis.

- Le niveau de section le plus élevé (`\part`) n'existe que pour les titres numérotés et à la particularité que le titre de la partie n'est pas imprimé (par défaut, mais cela peut être modifié). Cependant, même si le titre n'est pas imprimé, une page blanche est introduite (sur laquelle on peut supposer que le titre est imprimé une fois que l'utilisateur a reconfiguré la commande) et la numérotation de la *partie* est prise en compte pour calculer la numérotation des chapitres et autres sections.
La raison pour laquelle la version par défaut de `\part` n'imprime rien est que, selon le wiki ConTeXt presque toujours le titre à ce niveau nécessite une mise en page spécifique ; et bien que cela soit vrai, cela ne me semble pas une raison suffisante, puisque, dans la pratique, les chapitres et les sections sont aussi souvent redéfinis, et le fait que les parties n'impriment rien oblige l'utilisateur novice à *plonger* dans la documentation pour voir ce qui ne va pas.
- Bien que le premier niveau de sectionnement soit la « part », ceci n'est que théorique et abstrait. Dans un document spécifique, le premier niveau de sectionnement sera celui qui correspond à la première commande de sectionnement du document. C'est-à-dire que dans un document qui ne comprend pas de parties mais des chapitres, le chapitre sera le premier niveau. Mais si le document ne comprend pas non plus de chapitres, mais uniquement des sections, la hiérarchie de ce document commencera par les sections.

7.3 Syntaxe commune des commandes liées aux sections

Toutes les commandes de section, y compris les niveaux créés par l'utilisateur (voir [section 7.5](#)), permettent les formes alternatives de syntaxe suivantes (si, par exemple, nous utilisons le niveau « `section` ») :

```
\section [Label] {Title}
\section [Options]
\startsection [Options] [Variables] ... \stopsection
```

Dans les trois méthodes ci-dessus, les arguments entre crochets sont facultatifs et peuvent être omis. Nous les examinerons séparément, mais il convient tout d'abord de préciser que dans Mark IV, c'est la troisième de ces trois méthodes qui est recommandée.

- Dans la première forme syntaxique, que l'on pourrait appeler la « *historique* », la commande prend deux arguments, l'un facultatif entre crochets, l'autre obligatoire entre accolades. L'argument facultatif sert à associer la commande à une étiquette qui sera utilisée pour les références internes (voir [section ??](#)). L'argument obligatoire entre crochets est le titre de la section.
- Les deux autres formes de syntaxe sont plutôt du style de ConTeXt : tout ce que la commande doit savoir est communiqué par des valeurs et des options introduites entre crochets.

Rappelez-vous que dans [sections 3.3.1](#) et [3.4](#) j'ai dit que dans ConTeXt, la portée de la commande est indiquée entre crochets, et ses options entre crochets. Mais si l'on y réfléchit, le titre d'une commande de section particulière n'est pas le champ d'application de celle-ci, donc pour être cohérent avec la syntaxe générale, il ne devrait pas être introduit entre crochets, mais comme une option. ConTeXt permet cette exception car il s'agit de la façon historique de faire les choses dans T_EX, mais il fournit les formes alternatives de syntaxe qui sont plus cohérentes avec sa conception générale.

Les options sont du type affectation de valeur (OptionName=Value), et sont les suivantes :

- `reference` : étiquette, ou référence, pour les références croisées.
- `title` : titre de la section qui sera utilisé dans le corps du document.
- `list` : Le titre de la section qui sera utilisé dans la table des matières.
- `marking` : Le titre de la section qui sera utilisé dans les en-têtes ou les pieds de page.
- `bookmark` : Le titre de la section qui sera utilisé en *signet* dans le fichier PDF.
- `ownnumber` : Cette option est utilisée dans le cas d'une section qui n'est pas automatiquement numérotée ; dans ce cas, cette option prendra le numéro attribué à la section en question.

Bien entendu, les options « `list` », « `marking` » et « `bookmark` » ne doivent être utilisées que si nous voulons utiliser un titre différent pour remplacer le titre principal défini avec l'option « `title` ». Ceci est très utile, par exemple, lorsque le titre est trop long pour l'en-tête ; bien que pour y parvenir, nous puissions

également utiliser l'option `\nomarking` et `\nolist` (quelque chose de très similaire). D'autre part, nous devons garder à l'esprit que si le texte du titre (l'option « title ») comprend des virgules, il devra être placé entre accolades, à la fois le texte complet et la virgule, afin que ConTeXt sache que la virgule fait partie du titre. Il en va de même pour les options : « list », « marking » et « bookmark ». Par conséquent, pour ne pas avoir à surveiller s'il y a ou non des virgules dans le titre, je pense que c'est une bonne idée de prendre l'habitude de toujours enfermer la valeur de l'une de ces options entre des accolades.

Ainsi, par exemple, les lignes suivantes créeront un chapitre intitulé « Un Chapitre de test » associé à l'étiquette « chap :test » pour les références croisées, tandis que l'en-tête sera « Chapitre test » au lieu de « Un Chapitre de test ».

```
\chapter
[title={Un Chapitre de test},
 reference={chap:test},
 marking={Chapitre test}]
```

La syntaxe `\startSectionType` transforme la section en un *environnement*. Elle est plus cohérente avec le fait que, comme je l'ai dit au début, en arrière-plan, chaque section est un bloc de texte différencié, bien que ConTeXt, par défaut, ne considère pas les *environnements* générés par les commandes de section comme des *groupes*. Néanmoins, cette procédure est celle que Mark IV recommande, probablement parce que cette façon d'établir les sections nous oblige à indiquer expressément où commence et finit chaque section, ce qui facilite la cohérence de la structure et offre très probablement un meilleur support pour les sorties XML et EPUB. En fait, pour la sortie XML, c'est essentiel.

Lorsque nous utilisons `\startSection`, une ou plusieurs variables sont autorisées comme arguments entre crochets. Leur valeur peut ensuite être utilisée ultérieurement à d'autres endroits du document grâce à la commande `\structureUserVariable`.

```
\startSection[title={Mon joli
titre}]
Mon texte dans
\nameStructureVariable{section}{title}
\stopSection
```

1 Mon joli titre
Mon texte dans Mon joli titre

Le fait, pour l'utilisateur, de disposer ou accéder à des variables permet des utilisations très avancées de ConTeXt en raison du fait que des décisions peuvent être prises concernant la compilation ou non d'un fragment, ou de quelle manière le faire, ou avec quel modèle en fonction de la valeur d'une variable particulière. Ces utilitaires ConTeXt, cependant, dépassent le cadre du matériel que je souhaite traiter dans cette introduction.

7.4 Format et configuration des sections et de leurs titres

7.4.1 Les commandes `\setuphead` and `\setupheads`

Par défaut, ConTeXt affecte certaines caractéristiques à chaque niveau de section qui affectent principalement (mais pas uniquement) le format d'affichage du titre dans le corps principal du document, mais pas la manière dont le titre est affiché dans la table des matières ou les en-têtes et pieds de page. Nous pouvons modifier ces caractéristiques à l'aide de la commande `\setuphead`, dont la syntaxe est :

```
\setuphead [Sections] [Options]
```

où

- **Sections** fait référence au nom d'une ou plusieurs sections (séparées par des virgules) qui seront affectées par la commande. Cela peut être :
 - N'importe quelle section prédéfinie (partie, chapitre, titre, etc.), auquel cas on peut y faire référence soit par son nom, soit par son niveau. Pour les désigner par leur niveau, nous utilisons le mot « `section-NumLevel` », où `NumLevel` est le numéro de niveau de la section concernée. Ainsi, « `section-1` » est égal à « `part` », « `section-2` » est égal à « `chapter` », etc.
 - Tout type de section que nous avons nous-mêmes défini. À cet égard, voir [section 7.5](#).
- **Options** sont les options de configuration. Elles sont du type affectation explicite de valeur (`OptionName=valeur`). Le nombre d'options éligibles est très élevé (plus de soixante) et je vais donc les expliquer en les regroupant en catégories selon leur fonction. Je dois cependant préciser que je n'ai pas réussi à déterminer à quoi servent certaines de ces options ni comment elles sont utilisées. Je ne parlerai pas de ces options.

J'ai dit précédemment que `\setuphead` affecte les sections qui sont expressément indiquées. Mais cela ne signifie pas que la modification d'une section particulière ne doit en aucun cas affecter les autres sections, à moins qu'elles n'aient été expressément mentionnées dans la commande. En fait, c'est le contraire qui est vrai : la modification d'une section affecte les autres sections qui lui sont liées, même si cela n'a pas été explicité dans la commande. Le lien entre les différentes sections est de deux types :

- Les commandes non numérotées sont liées à la commande numérotée correspondante du même niveau, de sorte qu'un changement d'apparence de la commande numérotée affectera la commande non numérotée du même niveau ; mais pas l'inverse : le changement de la commande non numérotée n'affecte pas la commande numérotée. Cela signifie, par exemple, que si nous modifions un aspect de « `chapter` » (niveau 2), nous modifions également cet aspect dans « `titre` » ; mais la modification de « `title` » n'affectera pas « `chapter` ».

- Les commandes sont liées hiérarchiquement, de sorte que si nous modifions *certaines caractéristiques* dans un niveau particulier, la modification affectera tous les niveaux qui suivent. Cela ne se produit qu'avec certaines caractéristiques. La couleur, par exemple : si nous établissons que les sous-sections s'afficheront en rouge, nous changeons également les sous-sous-sections, les sous-sous-sections, etc. en rouge. Mais il n'en va pas de même avec d'autres caractéristiques, comme le style de police par exemple.

Avec la commande `\setupheads` ConTeXt fournit la commande `\setupheads` qui affecte globalement toutes les commandes de section. Le wiki ConTeXt indique, en référence à cette commande, que certaines personnes ont déclaré qu'elle ne fonctionnait pas. D'après mes tests, cette commande fonctionne pour certaines options mais pas pour d'autres. En particulier, elle ne fonctionne pas avec l'option « `style` », ce qui est frappant, puisque le style des titres est très probablement la chose que nous voudrions changer globalement pour qu'il affecte tous les titres. Mais cela fonctionne, d'après mes tests, avec d'autres options telles que, par exemple, « `number` » ou « `color` ». Ainsi, par exemple, `\setupheads[color=blue]` fera en sorte que tous les titres de notre document soient imprimés en bleu.

Étant donné que je suis un peu trop paresseux pour prendre la peine de tester chaque option pour voir si elle fonctionne ou non avec `\setupheads`. (rappelez-vous qu'il y en a plus de soixante), dans ce qui suit je ne ferai référence qu'à `\setuphead`.

Enfin, avant d'examiner les options spécifiques, nous devrions noter quelque chose qui est dit dans le wiki ConTeXt, bien que ce ne soit probablement pas dit au bon endroit : certaines options ne fonctionnent que si nous utilisons la syntaxe `\startSectionName`.

Cette information est contenue en relation avec `\setupheads`, mais pas avec `\setuphead` qui est pourtant l'endroit où la majeure partie des options sont expliquées et où, si cela ne doit être dit qu'à un seul endroit, cela il serait le plus raisonnable de l'indiquer. D'autre part, l'information ne mentionne que l'option « `insidesection` », sans préciser si cela se produit également avec d'autres options.

7.4.2 Parties du titre d'une section

Avant d'entrer dans les options spécifiques qui nous permettent de configurer l'apparence des titres, il convient de commencer par signaler qu'un titre de section peut comporter jusqu'à trois parties différentes, que ConTeXt nous permet de formater ensemble ou séparément. Ces éléments de titre sont les suivants :

- Le titre lui-même**, c'est-à-dire le texte qui le compose. En principe, ce titre est toujours affiché, sauf pour les sections du type « `part` » où le titre n'est pas affiché par défaut. L'option qui contrôle l'affichage ou non du titre est « `placehead` » dont les valeurs peuvent être « `yes` », « `no` », « `hidden` », « `empty` » ou « `section` ». La signification des deux premiers est claire. Mais je ne suis pas sûr des résultats des autres valeurs de cette option.

Par conséquent, si nous voulons que le titre soit affiché dans les sections de premier niveau, notre paramètre doit être le suivant :



```
\setuphead
[part]
[placehead=yes]
```

Le titre de certaines sections, comme nous le savons déjà, peut être envoyé automatiquement aux en-têtes et à la table des matières. En utilisant les options `list` et `marking` des commandes de section, on peut indiquer un autre titre à envoyer à la place. Il est également possible, lors de l'écriture du titre, d'utiliser les options `\nolist` ou `\nomarking` pour que certaines parties du titre soient remplacées par des ellipses dans la table des matières ou l'en-tête. Par exemple :

```
\startsection
[title={Influences of \nomarking{19th century} impressionism \nomarking{in
the 21st century}}]
```

écrira « Influences de ... l'impressionnisme ... » dans l'en-tête.

- **La numérotation.** Ce n'est le cas que pour les sections numérotées (partie, chapitre, section, sous-section...), mais pas pour les sections non numérotées (titre, sujet, sous-sujet). En fait, le fait qu'une section particulière soit numérotée ou non dépend des options « `number` » et « `incrementnumber` » dont les valeurs possibles sont « `yes` » et « `no` ». Dans les sections numérotées, ces deux options sont définies comme `yes` et dans les sections non numérotées, comme `no`.

Pourquoi y a-t-il deux options pour contrôler la même chose ? Parce qu'en fait, les deux options contrôlent deux choses différentes : l'une est de savoir si la section est numérotée ou non (`incrementnumber`) et l'autre est de savoir si le numéro est affiché ou non (`number`). Si `incrementnumber=yes` et `number=no` sont définis pour une section, nous obtiendrons une section non numérotée visuellement mais elle sera tout de même compilée par ConTeXt. Cela peut être utile pour inclure une telle section dans la table des matières, puisque d'ordinaire, celle-ci n'inclut que les sections numérotées. À cet égard, voir `sous-section ??` dans `section ??`.

- **Le libellé du titre.** En principe, cet élément des titres est vide. Mais nous pouvons lui associer une valeur, auquel cas, avant le numéro et le titre proprement dit, l'étiquette que nous avons attribuée à ce niveau sera affichée. Par exemple, dans les titres de chapitre, on peut vouloir que le mot « Chapitre » soit affiché, ou le mot « Partie » pour les parties. Pour ce faire, nous n'utilisons pas la commande `\setuphead` mais la commande `\setuplabeltext`, qui nous permet d'attribuer une valeur textuelle aux étiquettes des différents niveaux de sectionnement. Ainsi, par exemple, si nous voulons écrire « Chapitre » dans notre document avant les titres des chapitres, nous devons définir :

```
\setuplabeltext
[section=Section~]
\startsection
[title={Mon joli titre}]
Mon texte.
\stopsection
```

Section 1 Mon joli titre

Mon texte.

Dans l'exemple, après le nom attribué, j'ai inclus le caractère réservé « ~ » qui insère un espace blanc insécable après le mot. Si nous ne tenons pas à ce qu'un saut de ligne se produise entre l'étiquette et le numéro, nous pouvons simplement ajouter un espace blanc. Mais cet espace blanc (quel qu'il soit) est important ; sans lui, le numéro sera relié à l'étiquette et nous verrions, par exemple, « Section1 » au lieu de « Section 1 ».

7.4.3 Contrôle de la numérotation (dans les sections numérotées)

Nous savons déjà que les sections numérotées prédéfinies (part, chapter, section...) et le fait qu'une section particulière soit numérotée ou non, dépendent des options « `number` » et « `incrementnumber` » configurées avec `\setuphead`.

Par défaut, la numérotation des différents niveaux est automatique, sauf si nous avons attribué la valeur « `yes` » à l'option « `ownnumber` ». Lorsque « `ownnumber=yes` », il faut indiquer le numéro attribué à chaque commande. Cela se fait :

- Si la commande est invoquée en utilisant la syntaxe classique, en ajoutant un argument avec le numéro avant le texte du titre. Par exemple : `\chapter[13]{Titre titre du chapitre}` générera un chapitre auquel on a attribué manuellement le numéro 13.
- Si la commande a été invoquée avec la syntaxe spécifique à ConTeXt (`\Section-Type [Options]` ou `\startSectionType [Options]`), avec l'option « `ownnumber` ». Par exemple :
`\chapter[title={Titre du chapitre}, ownnumber=13]`, génère un chapitre auquel on a attribué manuellement le numéro 13.

Lorsque ConTeXt fait automatiquement la numérotation, il utilise des compteurs internes qui stockent les numéros des différents niveaux ; il y a donc un compteur pour les parties, un autre pour les chapitres, un autre pour les sections, etc. Chaque fois que ConTeXt trouve une commande de section, il effectue les actions suivantes :

- Elle augmente de « 1 » le compteur associé au niveau correspondant à cette commande.
- Elle remet à 0 les compteurs associés à tous les niveaux inférieurs à celui de la commande en question.

Cela signifie, par exemple, qu'à chaque fois qu'un nouveau chapitre est trouvé, le compteur de chapitre est augmenté de 1 et toutes les commandes de section, sous-section, sous-sous-section, etc. sont remises à 0 ; mais le compteur de parties n'est pas affecté.

Pour modifier le nombre à partir duquel le comptage doit commencer, utilisez la commande `\setupheadnumber` comme suit :

```
\setupheadnumber [SectionType] [Nombre à partir duquel compter]
```

où *Nombre à partir duquel compter* est le nombre à partir duquel les sections de tout type seront comptées. Ainsi, si *Nombre à partir duquel compter* est égal à zéro, la première section sera 1 ; s'il est égal à 10, la première section sera 11.

Cette commande nous permet également de modifier le modèle d'incrémentation automatique ; ainsi, nous pouvons, par exemple, faire en sorte que les chapitres ou les sections soient comptés par paires ou par trois. Ainsi, `\setupheadnumber[section][+5]` verra les chapitres numérotés 5 sur 5 ; et `\setupheadnumber[chapter][14, +5]` verra que le premier chapitre commence par 15 (14+1), le deuxième sera 20 (15+5), le troisième 25, etc.

A. Règles de remise à zéro des compteurs

Il est possible de définir des règles de comptage encore plus spécifique avec `\defineResetset`. La syntaxe ainsi que l'utilisation sont illustrées :

```
\defineResetset
[NomRègle]
[ . , . , . , . , ....]
[.]
\setupheads[sectionresetset=NomRègle]
```

Chaque point « . » prend la valeur soit 0, soit 1. La seconde paire de crochets accueille donc une liste de 1 et de 0. La première valeur est associée au premier niveau de section et ainsi de suite. Une valeur de 1 indique qu'une nouvelle section du niveau supérieur au niveau concerné doit s'accompagner d'une remise à zéro du niveau concerné. La troisième paire de crochet indique la valeur à utiliser par défaut. Ainsi voici un exemple définissant que l'on souhaite ne pas remettre à zéro le compteur du niveau section lorsqu'on change de chapitre :

```

\definersetset[norazsection][1,1,0,1][0]
\setupheads[sectionresetset=norazsection]
\setuphead[chapter][page=no]

\startchapter[title=chapter 1]
  \startsection[title=Section 1.1]
    \startsubsection[title=Section 1.1.1]
      \startsubsubsection[title=Section 1.1.1.1]
        \stopsubsubsection
      \startsubsubsection[title=Section 1.1.1.2]
        \stopsubsubsection
    \stopsection
  \startsection[title=Section 1.2]
    \startsubsection[title=Section 1.2.1]
      \stopsubsection
    \startsubsection[title=Section 1.2.2]
      \stopsubsection
  \stopsection
\stopchapter

\startchapter[title=chapter 2]
  \startsection[title=Section 2.1]
    \startsubsection[title=Section 2.1.1]
      \stopsubsection
    \startsubsection[title=Section 2.1.2]
      \stopsubsection
  \stopsection
  \startsection[title=Section 2.2]
    \startsubsection[title=Section 2.2.1]
      \stopsubsection
    \startsubsection[title=Section 2.2.2]
      \stopsubsection
  \stopsection
\stopchapter

```

1 chapter 1

1.1 Section 1.1

1.1.1 Section 1.1.1

1.1.2 Section 1.1.2

1.2 Section 1.2

1.2.1 Section 1.2.1

1.2.2 Section 1.2.2

2 chapter 2

2.3 Section 2.1

2.3.1 Section 2.1.1

2.3.2 Section 2.1.2

2.4 Section 2.2

2.4.1 Section 2.2.1

2.4.2 Section 2.2.2

B. Règles de numérotation

Par défaut, la numérotation des sections affiche des chiffres arabes, et la numérotation de tous les niveaux précédents est incluse. Autrement dit, dans un document comportant des parties, des chapitres, des sections et des sous-sections, une sous-section spécifique indiquera à quelle partie, chapitre et section elle correspond. Ainsi, la quatrième sous-section de la deuxième section du troisième chapitre de la première partie sera « 1.3.2.4 ».

Les deux options de base permettant de contrôler l'affichage des nombres sont les suivantes :

- conversion** : Il contrôle le type de numérotation qui sera utilisé. Il autorise de nombreuses valeurs en fonction du type de numérotation que l'on souhaite :
 - **Numérotation avec chiffres arabes** : La numérotation classique : 1, 2, 3, ... est obtenue avec les valeurs `n`, `N` ou `numbers`.
 - **Numérotation avec des chiffres romains**. Trois façons de procéder :
 - * chiffres romains majuscules : `I`, `R`, `RomanNumerals`.
 - * chiffres romains minuscules : `i`, `r`, `romanNumerals`.
 - * chiffres romains en petites majuscules : `KR`, `RK`.
 - **Numérotation avec des lettres**. Trois façons de procéder :
 - * lettres majuscules : `A`, `Caractère`
 - * lettres minuscules : `a`, `character`
 - * lettres en petites majuscules : `AK`, `KA`
 - **Numérotation en mots**. C'est-à-dire qu'on écrit le mot qui désigne le nombre. Ainsi, par exemple, « 3 » devient « Trois ». Il y a deux façons de procéder :
 - * mots commençant par une majuscule : `Words`.
 - * mots en minuscule : `mots`.
 - **Numérotation par symboles** : La numérotation avec symboles utilise différents jeux de symboles dans lesquels une valeur numérique est attribuée à chaque symbole. Comme les jeux de symboles utilisés par ConTeXt ont un nombre très limité, il n'est approprié d'utiliser ce type de numérotation que lorsque le nombre maximal à atteindre n'est pas trop élevé. ConTeXt prévoit quatre jeux de symboles différents : `set 0`, `set 1`, `set 2` et `set 3` respectivement. Vous trouverez ci-dessous les symboles que chacun de ces jeux utilise pour la numérotation. Notez que le nombre maximum qui peut être atteint est de 9 dans les jeux de symboles `set 0` et `set 1` et de 12 dans les jeux de symboles `set 2` et `set 3` :

```

Set 0:• - * ▷ ° ○ ○ □ ✓
Set 1:★ ** *** ♫ §§ $$$ * ** ***
Set 2:★ + ♫ ** ++ §§ *** +++ §§ **** +++++ §§§§§
Set 3:★ ** *** ♫ §§ $$$ ¶ ¶¶ ¶¶¶ § §§ §§§

```

- sectionsegments** : Cette option nous permet de contrôler l'affichage ou non de la numérotation des niveaux précédents. Nous pouvons indiquer quels niveaux précédents seront affichés. Cela se fait en identifiant le niveau initial et le niveau final à afficher. L'identification du niveau peut se faire par son numéro (`partie=1`, `chapitre=2`, `section=3`, etc.), ou par son nom (`partie`, `chapitre`, `section`, etc.).

Ainsi, par exemple, « `sectionsegments=2:3` » indique que la numérotation des chapitres et des sections doit être affichée. C'est exactement la même chose que de dire « `sectionsegments=chapitre:section` ». Si nous voulons indiquer que tous les numéros supérieurs à un certain niveau sont affichés, nous pouvons utiliser, comme valeur de « `optionsegments` » `Initial Level :all`, ou `InitialLevel :*`. Par exemple, « `sectionsegments=3:*` » indique que la numérotation est affichée à partir du niveau 3 (section).

Ainsi, par exemple, imaginons que nous voulons que les parties soient numérotées en chiffres romains en lettres majuscules ; les chapitres en chiffres arabes, mais sans inclure le numéro de la partie à laquelle ils appartiennent ; les sections et sous-sections en chiffres arabes incluant les numéros de chapitre et de section, et les sous-sections en lettres majuscules. Il faut écrire ce qui suit :

```
\setuphead [part]           [conversion=I]
\setuphead [chapter]        [conversion=n, sectionsegments=1:2]
\setuphead [section]         [conversion=r, sectionsegments=2:3]
\setuphead [subsection]      [conversion=a, sectionsegments=2:4]
\setuphead [subsubsection]    [conversion=KA, sectionsegments=5]

\startpart                 [title=Ma partie]
\startchapter               [title=Mon chapitre]
\startsection               [title=Ma section]
\startsubsection             [title=Ma sous-section]
\startsubsubsection          [title=Ma sous-sous-section]
texte
\stopsubsubsection
\stopsubsection
\stopsection
\stopchapter
\stoppart
```

1.1 Mon chapitre

i.i Ma section

a.a.a Ma sous-section

A Ma sous-sous-section

texte

Nous voyons dans ce dernier exemple que l'option `conversion` affecte l'ensemble des éléments qui composent la numérotation d'un niveau de section. Bien souvent nous préférons maintenir une cohérence d'un niveau de section à l'autre pour faciliter la lecture. Pour cela nous utiliserons `\definestructureconversionset`.

```

\definestructureconversionset
[mysectionnumbers]
[I,n,r,a,KA]
[n]

\setupheads[sectionconversionset=mysectionnumbers]

\setuphead [chapter]      [sectionsegments=1:2]
\setuphead [section]       [sectionsegments=2:3]
\setuphead [subsection]    [sectionsegments=2:4]
\setuphead [subsubsection] [sectionsegments=5]

\startpart      [title=Ma partie]
\startchapter   [title=Mon chapitre]
\startsection   [title=Ma section]
\startsubsection [title=Ma sous-section]
\startsubsubsection [title=Ma sous-sous-section]
\starttexte
\stopsubsubsection
\stopsubsection
\stopsection
\stopchapter
\stoppart

```

I.1 Mon chapitre

1.i Ma section

1.i.a Ma sous-section

A Ma sous-sous-section

texte

Il devient alors plus facile pour le lecteur de repérer que le 1 en numérotation arabe fait référence à la numérotation du niveau chapitre et ainsi de suite.

C. Séparateurs

Profitons-en pour introduire ici deux compléments de configuration qui sont les symboles utilisés en tant que séparateur de chaque élément de la numérotation, avec `\definestructureseparatorset` (donc la syntaxe est proche de celle précédemment vue pour `\definestructureconversionset`), et le symbole utilisé comme séparateur entre la numérotation et le texte du titre, avec l'option `sectionstopper`

```

\startchapter [title=Section 1]
  \startsection [title=Section 1.1]
    \startsubsection [title=Section 1.1.1]
      \stopsubsection
    \stopsection
  \stopchapter

\definestructureseparatorset
  [default]
  [{.},{/},{-}]
  [{/}]

\setupheads
  [sectionstopper={}]
```

```

\startchapter [title=Section 2]
  \startsection [title=Section 2.1]
    \startsubsection [title=Section 2.1.1]
      \stopsubsection
    \stopsection
  \stopchapter
```

1 Section 1

1.1 Section 1.1

1.1.1 Section 1.1.1

2) Section 2

2/1) Section 2.1

2/1-1) Section 2.1.1

D. Raffinements

Voyons ici une dernière illustration de ce que permet ConTeXt sur le mécanisme de numérotation des sections. Imaginons que nous souhaitons, pour facilier encore la lecture, faire varier la taille de police de la numérotation en fonction du niveau de section. Nous allons personnaliser avec la commande `\defineconversion`

```

\def\NiveauUn#1{\tf{#1}}
\def\NiveauDeux#1{\fc{#1}}
\def\NiveauTrois#1{\fb{#1}}
\def\NiveauQuatre#1{\fa{#1}}
```

```

\defineconversion[N1][\NiveauUn]
\defineconversion[N2][\NiveauDeux]
\defineconversion[N3][\NiveauTrois]
\defineconversion[N4][\NiveauQuatre]
```

```

\definestructureconversionset
  [mysectionnumbers]
  [N1,N2,N3,N4]
  [N]
\setupheads
  [sectionconversionset=mysectionnumbers]
```

```

\startpart[title=Partie 1]
  \startchapter[title=Chapitre 1.1]
    \startsection[title=Section 1.1.1]
      \startsubsection[title=Section 1.1.1.1]
        \stopsubsection
      \stopsection
    \stopchapter
  \stoppart
```

1.1 Chapitre 1.1

1.1.1 Section 1.1.1

1.1.1.1 Section 1.1.1.1

7.4.4 Couleur et style du titre

Nous disposons des options suivantes pour contrôler le style et la couleur :

- **Le style** est contrôlé par les options « `style` », « `numberstyle` » et « `textstyle` » selon que l'on souhaite affecter l'ensemble du titre, uniquement la numérotation ou uniquement le texte. Au moyen de l'une de ces options, nous pouvons inclure des commandes qui affectent la police, à savoir : police spécifique, style (romain, sans serif ou à chasse fixe), alternative (italique, gras, oblique...) et taille. Si l'on veut indiquer une seule caractéristique de style, on peut le faire soit en utilisant le nom du style (par exemple, « `bold` » pour bold), soit en indiquant son abréviation (« `bf` »), soit la commande qui la génère (`\bf`, dans le cas de bold). Si nous voulons indiquer plusieurs caractéristiques simultanément, nous devons le faire au moyen des commandes qui les génèrent, en les écrivant les unes après les autres. N'oubliez pas, par ailleurs, que si vous n'indiquez qu'une seule caractéristique, le reste des caractéristiques de style sera établi automatiquement avec les valeurs par défaut du document, c'est pourquoi il est rarement conseillé d'établir une seule caractéristique de style.
- **La couleur** est définie avec les options « `color` », « `numbercolor` » et « `textcolor` » selon que l'on souhaite définir la couleur de l'ensemble du titre, ou seulement la couleur de la numérotation ou du texte. La couleur indiquée ici peut être l'une des couleurs prédéfinies de ConTeXt, ou une autre couleur que nous avons définie nous-mêmes et à laquelle nous avons préalablement attribué un nom. Cependant, nous ne pouvons pas utiliser directement une commande de définition de couleur ici.

En plus de ces six options, il existe encore cinq autres options permettant de mettre en place des fonctionnalités plus sophistiquées avec lesquelles nous pouvons faire pratiquement tout ce que nous voulons. Il s'agit de : « `command` », « `numbercommand` », « `textcommand` », « `deepnumbercommand` » et « `deeptextcommand` ». Commençons par expliquer les trois premières :

- **command** indique une commande qui prendra deux arguments, le numéro et le titre de la section. Il peut s'agir d'une commande ConTeXt normale ou d'une commande que nous avons définie nous-mêmes.
- **numbercommand** est similaire à « `command` », mais cette commande ne prend en argument que le numéro de la section.
- **textcommand** est également similaire à « `command` », mais elle ne prend en argument que le texte du titre.

Ces trois options nous permettent de faire pratiquement tout ce que nous voulons. Par exemple, si je veux que les sections soient alignées à droite, enfermées dans un cadre et avec un retour à la ligne entre le numéro et le texte, je peux simplement créer une commande à cet effet, puis indiquer cette commande comme valeur de la commande « `command` ». Cela pourrait être réalisé avec les lignes suivantes :

```

\startsection[title=Joli titre ici]
Mon texte.
\stopsection

\define[2]\AlignSection
{\framed
 [frame=on,
 width=broad,
 align=flushright]{#1\#2}]

\setuphead
[section]
[style=bold,
 color=darkred,
 command=\AlignSection]

\startsection[title=Joli titre là]
Mon texte.
\stopsection

```

1 Joli titre ici

Mon texte.

2
Joli titre là

Mon texte.

Lorsque nous définissons simultanément les options « `command` » et « `style` », la commande est appliquée au titre avec son style. Cela signifie, par exemple, que si nous avons défini « `style de texte=\em` », et « `commande de texte=\WORD` », la commande `\WORD` (qui met en majuscule le texte qu'elle prend comme argument) sera appliquée au titre avec son style, c'est-à-dire : `\WORD{\em Texte du titre}`. Si nous voulons que cela se fasse dans l'autre sens, c'est-à-dire que le style soit appliqué au contenu. Si l'on veut que le style soit appliqué au contenu du titre une fois la commande appliquée, il faut utiliser, au lieu des options « `commandtext` » et « `commandnumber` », les options « `commanddeeptext` » et « `commanddeepnumber` ». Ainsi, dans l'exemple donné ci-dessus, on obtiendrait « `{\em\WORD{Texte du titre}}` ».

Dans la plupart des cas, il n'y a pas de différence entre les deux méthodes. Mais dans certains cas, il peut y en avoir une.

7.4.5 Emplacement du numéro et du texte du titre

L'option « `alternative` » contrôle deux choses simultanément : l'emplacement de la numérotation par rapport au texte du titre, et l'emplacement du titre lui-même (y compris le numéro et le texte) par rapport à la page sur laquelle il est affiché et au contenu de la section. Ce sont deux choses différentes, mais comme elles sont régies par la même option, elles sont contrôlées simultanément.

L'emplacement du titre par rapport à la page et au premier paragraphe du contenu de la section est contrôlé par les valeurs possibles suivantes de « `alternative` » :

- **`text`** : Le titre de la section est intégré au premier paragraphe de son contenu. L'effet est similaire à celui produit dans L^AT_EX avec `\paragraph` et `\subparagraph`.
- **`paragraph`** : Le titre de la section sera un paragraphe indépendant.

- **normal** : Le titre de la section sera placé à l'emplacement par défaut fourni par ConTeXt pour le type particulier de section en question. Normalement, il s'agit de « `paragraph` ».
- **middle** : Le titre est écrit comme un paragraphe autonome, centré. S'il s'agit d'une commande numérotée, le numéro et le texte sont séparés sur des lignes différentes, toutes deux centrées.
Un effet similaire à celui obtenu avec « `alternative=middle` » est obtenu avec l'option « `align` » qui contrôle l'alignement du titre. Elle peut prendre les valeurs « `left` », « `middle` » ou « `flushright` ». Mais si nous centrons le titre avec cette option, le numéro et le texte apparaîtront sur la même ligne.
- **margintext** : Cette option permet d'imprimer la totalité du titre (numérotation et texte) dans l'espace réservé à la marge.

	1	
	1 Joli titre <code>text</code> Mon texte.	
	2 Joli titre <code>paragraph=normal</code> Mon texte.	
	3	
	Joli titre <code>middle</code> Mon texte.	
4 Joli titre mar- gintext	Mon texte.	



L'emplacement du numéro par rapport au texte du titre est indiqué par les valeurs possibles suivantes de « `alternative` » :

- **margin/inmargin** : Le titre constitue un paragraphe distinct. La numérotation est écrite dans l'espace réservé à la marge. Je n'ai pas encore compris la différence entre l'utilisation de « `margin` » et l'utilisation de « `inmargin` ».
- **reverse** : Le titre constitue un paragraphe distinct, mais l'ordre normal est inversé, et le texte est imprimé en premier, puis le numéro.
- **top/bottom** : Dans les titres dont le texte occupe plus d'une ligne, ces deux options permettent de contrôler si la numérotation sera alignée sur la première ligne du titre ou sur la dernière ligne respectivement.

		1	
1	Joli titre margin Mon texte. Joli titre reverse 2 Mon texte. Joli titre 3 bottom Mon texte. 4 Joli titre top Mon texte.		

7.4.6 Commandes ou actions à effectuer avant ou après un titre ou une section

Il est possible d'indiquer une ou plusieurs commandes qui sont exécutées avant l'affichage du titre (option « `before` ») ou après (option « `after` »). Ces options sont largement utilisées pour marquer visuellement le titre. Par exemple, si nous voulons ajouter un espace vertical supplémentaire entre le titre et le texte qui le précède, l'option « `before=\blank` » ajoutera une ligne blanche. Pour ajouter encore plus d'espace, nous pourrions écrire « `before={\blank[3*big]}` ». Dans ce cas, nous avons entouré la valeur de l'option de crochets pour éviter une erreur. Nous pourrions également indiquer visuellement la distance entre le texte précédent et le suivant avec « `before=\hairline, after=\hairline` », qui tracerait une ligne horizontale avant et après le titre.



Très similaires aux options « `beforesection` » et « `aftersection` » sont les options « `commandbefore` » et « `commandafter` ». D'après mes tests, je déduis que la différence est que les deux premières options exécutent des actions avant et après le début de la composition du titre en tant que tel, tandis que les deux dernières font référence à des commandes qui seront exécutées avant et après la composition du *texte du titre*.

Dans la même veine, les options « `beforesection` » et « `aftersection` » permettent d'indiquer des commandes à exécuter respectivement avant et après l'ensemble de la section, c'est à dire avant l'affichage du titre et après le dernier mots de la section (c'est à dire lors du `\stopsection`).

Si, au lieu des commandes de section, nous utilisons les environnements de section (`\start ... \stop`), nous disposons également de l'option « `insidesection` », grâce à laquelle nous pouvons indiquer une ou plusieurs commandes qui seront exécutées une fois que le titre aura été composé et que nous serons déjà à l'intérieur de la section. Cette option nous permet, par exemple, de nous assurer qu'immédiatement après le début d'un chapitre, une table des matières sera automatiquement tapée avec (« `insidesection=\placecontent` »).

```
\setuphead
[section]
[beforesection={1-beforesection},
 aftersection={2-aftersection},
 before={3-before},
 after={4-after},
 inbetween={5-inbetween},
 insidesection={6-insidesection},
 style=bold]%
\startsection[title={Joli titre}]
Mon texte.
\stopsection
```

1-beforesection
 3-before5-inbetween1 Joli titre
 4-after6-insidesection Mon texte. 2-aftersection

Si l'on veut insérer un saut de page avant le titre, il faut utiliser l'option « `page` » qui permet, entre autres valeurs, « `yes` » pour insérer un saut de page, « `left` » pour insérer autant de sauts de page que nécessaire pour que le titre commence sur une page paire, « `right` » pour que le titre commence sur une page impaire, ou « `no` » si l'on veut désactiver le saut de page forcé. Par contre, pour les niveaux inférieurs à « `chapter` », cette option ne fonctionnera que si la « `continue=no` » est utilisée, sinon elle ne fonctionnera pas si la section, la sous-section ou la commande se trouve sur la première page d'un chapitre.

Par défaut, les chapitres commencent sur une nouvelle page dans ConTeXt. S'il est établi que les sections commencent aussi sur une nouvelle page, le problème se pose de savoir ce qu'il faut faire avec la première section d'un chapitre qui, peut-être, se trouve au début du chapitre : si cette section commence aussi un saut de page, on se retrouve avec la page qui ouvre le chapitre ne contenant que le titre du chapitre, ce qui n'est pas très esthétique. C'est pourquoi on peut définir l'option « `continue` », un nom, je dois dire, qui n'est pas très clair pour moi : si « `continue=yes` », le saut de page ne s'appliquera pas aux sections qui sont sur la première page d'un chapitre. Si « `continue=no` », le saut de page sera quand même appliqué.

7.4.7 Autres fonctionnalités configurables

En plus de celles que nous avons déjà vues, nous pouvons configurer les fonctionnalités supplémentaires suivantes avec `\setuphead` :

- **Interligne.** Contrôlé par la « `interlinespace` » qui prend comme valeur le nom d'une commande interligne préalablement créée avec `\defineinterlinespace` et configurée avec `\setupinterlinespace`.
- **Alignement.** L'option « `align` » affecte l'alignement du paragraphe contenant le titre. Elle peut prendre, entre autres, les valeurs suivantes : « `flushleft` » (gauche), « `flushright` » (droite), « `middle` » (centré), « `inner` » (marge intérieure) et « `outer` » (marge extérieure).

- **Marge.** L'option « `margin` » permet de définir manuellement la marge du titre.
- **Indentation du premier paragraphe.** La valeur de l'option « `indentnext` » (qui peut être "yes", "no" ou "auto") contrôle si la première ligne du premier paragraphe de la section sera mise en retrait ou non. Le fait qu'elle soit ou non en retrait (dans un document où la première ligne des paragraphes est généralement en retrait) est une question de goût.
- **Largeur.** Par défaut, les titres occupent la largeur dont ils ont besoin, sauf si celle-ci est supérieure à la largeur de la ligne, auquel cas le titre occupera plus d'une ligne. Mais avec l'option « `width` », nous pouvons attribuer une largeur particulière au titre. Les options « `numberwidth` » et « `textwidth` » permettent respectivement d'affecter la largeur de la numérotation ou la largeur du texte du titre.
- **Séparation du numéro et du texte.** Les options « `distance` » et « `textdistance` » permettent de contrôler la distance séparant le numéro du titre du texte du titre.
- **Style des en-têtes et des pieds de section.** Pour cela, nous utilisons les options « `header` » et « `footer` ».

7.4.8 Autres options de `\setuphead`

Avec les options que nous avons déjà vues, nous pouvons constater que les possibilités de configuration des titres de section sont presque illimitées. Cependant, `\setuphead` possède une trentaine d'options que je n'ai pas mentionnées. La plupart parce que je n'ai pas découvert à quoi elles servent ou comment elles sont utilisées, quelques-unes parce que leur explication m'obligerait à entrer dans des aspects que je n'ai pas l'intention de traiter dans cette introduction.



7.5 Définir de nouvelles commandes de section

Nous pouvons définir nos propres commandes de section avec `\definehead` dont la syntaxe est :

```
\definehead[NomCommande][Modèle][Configuration]
```

où

- **NomCommande** représente le nom que portera la nouvelle commande de section.
- **Modèle** est le nom d'une commande de section existante qui sera utilisée comme modèle à partir duquel la nouvelle commande héritera initialement de toutes ses caractéristiques.
En fait, la nouvelle commande hérite du modèle bien plus que ses caractéristiques initiales : elle devient une sorte d'instance personnalisée du modèle, mais partage avec lui, par exemple, le compteur interne qui contrôle la numérotation.
- **Configuration** est la configuration personnalisée de notre nouvelle commande. Ici, nous pouvons utiliser exactement les mêmes options que dans `\setuphead`.

Il n'est pas nécessaire de configurer la nouvelle commande au moment de sa création. Cela peut être fait plus tard avec `\setuphead` et, en fait, dans les exemples donnés dans les manuels ConTeXt et son wiki, cela semble être la manière habituelle.

7.6 La macrostructure du document

Les chapitres, sections, sous-sections, titres..., structurent le document ; ils l'organisent. Mais parallèlement à la structure résultant de ce type de commandes, il existe dans certains livres imprimés, notamment ceux issus du monde académique, un *macro-ordonnancement* du matériel du livre, qui tient compte non pas de son contenu mais de la fonction que chacune de ces grandes parties remplit dans le livre. C'est ainsi que l'on fait la différence entre :

- **Pages initiales ou préliminaires**, contenant la couverture, la page de titre, la page de remerciements, une page de dédicace, la table des matières, éventuellement une préface, un prologue, une page de présentation, etc.
- **Le corps principal** du document, qui contient le texte fondamental du document, divisé en parties, chapitres, sections, sous-sections, etc. Cette partie est généralement la plus étendue et la plus importante.
- **Annexes** composé de contenus complémentaires qui développent ou illustrent une question traitée dans le corps du document, ou fournissent une documentation supplémentaire non rédigée par l'auteur du corps du document, etc.
- **Pages finales** du document où l'on trouve habituellement l'épilogue, la bibliographie, des index, le glossaire, le colophon etc.

Dans le fichier source, nous pouvons délimiter chacune de ces macro-sections (ou blocs) grâce aux environnements vus dans la [table 7.2](#).

Macro-section	Nom ConTeXt Commande
Pages préliminaires	frontpart
Corps principal	bodypart
Annexes	appendix
Pages finales	backpart

Tableau 7.2 Environnements qui reflètent la macrostructure du document

Les quatre environnements permettent les quatre mêmes options : « `page` », « `before` », « `after` » et « `number` », et leurs valeurs et utilité sont les mêmes que celles trouvées dans `\setuphead` (voir [section 7.4](#)), bien que nous devons noter qu'ici l'option « `number=no` » éliminera la numérotation de toutes les commandes de sectionnement dans l'environnement.

L'inclusion de l'une de ces macro-sections (ou bloc) dans notre document n'a de sens que si elle permet d'établir une certaine différenciation entre elles. Il peut s'agir d'en-têtes ou d'une numérotation de pages dans le corps du texte. La configuration de chacun de ces blocs est réalisée par `\setupsectionblock` dont la syntaxe est :

```
\setupsectionblock [NomMacroSection] [Options]
```

où `NomMacroSection` peut être `frontpart`, `bodypart`, `appendix` ou `backpart` et les options peuvent être les mêmes que celles mentionnées précédemment : « `page` », « `number` », « `before` » et « `after` ».

ConTeXt permet d'attribuer des configurations différentes aux commandes selon que l'on fait appel à elles dans telle ou telle macro-section (ou bloc). La fonction pour cela est `\startsectionblockenvironment`. Ainsi, par exemple, pour s'assurer que dans *frontmatter* les pages sont numérotées avec des chiffres romains, nous devrions écrire dans le préambule de notre document :

```
\startsectionblockenvironment [frontpart]
\setupuserpagenumber
    [numberconversion=romannumerals]
\stopsectionblockenvironment
```



Les configurations par défaut de ces quatre macro-sections (ou bloc) impliquent notamment que :

- Les quatre macro-section commencent une nouvelle page.
- La numérotation des sections dépend de la macro-section :
 - Dans *frontmatter* et *backmatter* toutes les sections numérotées deviennent non numérotées par défaut.
 - Dans *bodymatter* les sections ont une numérotation arabe.
 - Dans *appendices*, les sections sont numérotées en majuscules.

Il est également possible de créer de nouveaux macro-section avec `\definesectionblock`.

Chapitre 8

Table des matières, index, listes

Table of Contents : 8.1 Table des matières ; 8.1.1 Vue d'ensemble de la table des matières ; 8.1.2 Table des matières entièrement automatique avec un titre ; 8.1.3 Table des matières automatique sans titre ; 8.1.4 Option de sélection des éléments intégrés à la TdM : l'option **criterium** ; 8.1.5 Mise en page de la TdM : l'option **alternative** ; 8.1.6 Format des entrées de la TdM ; 8.1.7 Ajustements manuels ; A Inclure les sections non numérotées dans la TdM ; B Ajouter des entrées manuellement à la TdM ; C Exclure de la TdM une section particulière appartenant à un type de section qui est inclus dans le TdM ; D Textes du titre différents entre TdM et corps du document ; 8.2 Listes, listes combinées et tables des matières basées sur une liste ; 8.2.1 Listes de ConTeXt ; 8.2.2 Listes ou index d'images, de tableaux et d'autres éléments ; 8.2.3 Listes combinées ; 8.3 Index ; 8.3.1 Génération de l'index ; A The prior definition of the entries in the index and the marking of the points in the source file that refer to them ; B Generating the final index ; 8.3.2 Formatage de l'index ; 8.3.3 Création d'autres index ;

Une table des matières et un index sont des éléments globaux d'un document. Presque tous les documents auront une table des matières, tandis que seuls certains documents auront un index. Dans de nombreuses langues (mais pas en anglais), la table des matières et l'index sont regroupés sous le terme général « index ». Pour les lecteurs anglais, la table des matières se trouve normalement au début (d'un document, voire dans certains cas au début des chapitres), et l'index à la fin.

L'un et l'autre impliquent une application particulière du mécanisme des références internes dont l'explication est incluse dans [section 9.2](#).

8.1 Table des matières

8.1.1 Vue d'ensemble de la table des matières

Dans le chapitre précédent, nous avons examiné les commandes qui permettent d'établir la structure d'un document au fur et à mesure de sa rédaction. Cette section se concentre sur la table des matières et l'index, qui reflètent en quelque sorte la structure du document et offrent des outils efficaces pour y naviguer. La table

des matières est très utile pour se faire une idée du document dans son ensemble (elle permet de contextualiser l'information) et pour rechercher l'endroit exact où se trouve un passage particulier. Les livres dont la structure est très complexe, avec de nombreuses sections et sous-sections de différents niveaux de profondeur, semblent nécessiter un autre type de table des matières, car une table des matières moins détaillée (peut-être avec seulement les deux ou trois premiers niveaux de sectionnement) est essentielle pour se faire une idée générale du contenu du document. Par contre, elle ne permet pas de localiser un passage particulier ce qui peut être l'objet d'une table des matières très détaillée. Cette dernière ayant pour faiblesse de facilement faire perdre la vue d'ensemble à son lecteur. C'est pourquoi, parfois, les livres dont la structure est particulièrement complexe comportent plusieurs tables des matières : une table des matières peu détaillée en début d'ouvrage, indiquant les parties principales, une table des matières plus détaillée au début de chaque chapitre ainsi que, éventuellement, un index en fin de document.

Tous ces éléments peuvent être générés automatiquement par ConTeXt avec une relative facilité. On peut :

- Générer une table des matières complète ou partielle à n'importe quel endroit du document.
- Décider du contenu de l'une ou l'autre.
- Configurer leur apparence jusqu'au moindre détail.
- Inclure des hyperliens qui nous permettent de passer directement à la section en question.

En fait, cette dernière fonctionnalité est incluse par défaut dans toutes les tables des matières, à condition que la fonction d'interactivité ait été activée dans le document. Voir, à ce sujet, [section 9.3](#).

L'explication de ce mécanisme dans le manuel de référence de ConTeXt est, à mon avis, quelque peu confuse, ce qui, je pense, est dû au fait que trop d'informations sont introduites en même temps. Le mécanisme de construction des tables des matières de ConTeXt comporte de nombreux éléments, et il est difficile de les expliquer tous tout en restant clair. En revanche, l'explication dans [wiki](#), se limite pratiquement des exemples : cela est très utile pour apprendre les *trucs* mais inadéquat – je pense – pour comprendre le mécanisme et comment il fonctionne. C'est pourquoi la stratégie que j'ai décidé d'utiliser pour expliquer les choses dans cette introduction commence par supposer quelque chose qui n'est pas strictement vrai (ou pas complètement vrai) : qu'il existe quelque chose dans ConTeXt appelé la *table des matières*. A partir de là, les commandes *normales* pour générer la table des matières sont expliquées, et quand ces commandes et leur configuration sont bien connues, je pense que c'est le moment d'introduire – bien qu'à un niveau théorique plutôt que plus pratique – les informations sur les pièces du mécanisme qui ont été omises jusqu'alors. La connaissance de ces *éléments supplémentaires* nous permet de créer des tables des matières beaucoup plus personnalisées que celles que nous pouvons appeler les *éléments de base* créées avec les commandes expliquées jusqu'à ce point ; cependant, dans la plupart des cas, nous n'aurons pas besoin de le faire.

8.1.2 Table des matières entièrement automatique avec un titre

Les commandes de base pour générer automatiquement une table des matières (TdM ou ToC en anglais) à partir des sections numérotées d'un document (`part`, `chapter`, `section`, etc.) sont `\completecontent` et `\placecontent`. La principale différence entre ces deux commandes est que la première ajoute un *titre* à la Table des matières ; pour ce faire, elle insère immédiatement avant la Table des matières un *chapitre non numéroté* dont le titre par défaut est « Table des matières ».

Par conséquent, `\completecontent` :

- Insère, à l'endroit où il se trouve, un nouveau chapitre non numéroté intitulé « Table des matières ».
Nous rappelons que dans ConTeXt la commande utilisée pour générer une section non numérotée au même niveau que les chapitres, est `\title`. (voir [section 7.2](#)). Par conséquent, en réalité, `\completecontent` n'insère pas un *Chapitre* (`\chapter`) mais un *Title* (`\title`). Je ne l'ai pas dit parce que je pense qu'il peut être déroutant pour le lecteur d'utiliser les noms des commandes de section non numérotées ici, puisque le terme *Title* a également un sens plus large, et il est facile pour le lecteur de ne pas l'identifier avec le niveau concret de section auquel nous faisons référence.
- Ce *chapitre* (en fait, `\title`) est formaté exactement de la même manière que le reste des chapitres non numérotés du document, ce qui inclut par défaut un saut de page.
- La table des matières est affichée immédiatement après le titre.

Initialement, la TdM générée est *complète*, comme nous pouvons le déduire du nom de la commande qui la génère (`\completecontent`). Mais d'une part, nous pouvons limiter le niveau de profondeur de la TdM comme expliqué dans [section 8.1.3](#) et, d'autre part, comme cette commande est *sensible* à l'endroit où elle se trouve dans le fichier source (voir ce qui est dit plus loin à propos de `\placecontent`), si `\completecontent` ne se trouve pas au début du document, il est possible que la TdM générée ne soit pas complète ; et à certains endroits du fichier source, il est même possible que la commande soit apparemment ignorée. Si cela se produit, la solution consiste à invoquer la commande avec l'option « `criterium=all` ». À propos de cette option, voir également [section 8.1.3](#).

Pour modifier le titre par défaut attribué aux TdM, nous utilisons la commande `\setupheadtext` dont la syntaxe est :

```
\setupheadtext [Langage] [Élément=Texte]
```

où *Langage* est facultatif et fait référence à l'identificateur de langue utilisé par ConTeXt (voir [section 10.5](#)), et *Élément* fait référence à l'élément dont nous voulons modifier le nom (« *content* » dans le cas de la table des matières) et *Texte* est le texte du titre que nous voulons donner à notre TdM. Par exemple

```
\setupheadtext [fr] [content={Carte de navigation}]
```

fera en sorte que la TdM générée par `\completecontent` soit intitulée « Carte de navigation » au lieu de « Table des matières ».

De plus, `\completecontent` permet les mêmes options de configuration que `\placecontent`, pour l'explication desquelles je renvoie à la section suivante.

8.1.3 Table des matières automatique sans titre

La commande générale d'insertion d'une TdM sans titre, générée automatiquement à partir des commandes de sectionnement du document, est `\placecontent`, dont la syntaxe est :

```
\placecontent [Options]
```

En principe, la table des matières contiendra absolument toutes les sections numérotées, bien que nous puissions limiter son niveau de profondeur avec la commande `\setupcombinedlist` (dont nous parlerons plus loin). Ainsi, par exemple :

```
\setupcombinedlist [content] [list={chapter,section}]
```

limitera le contenu de la table des matières aux deux niveaux indiqués : chapitre et section.

Une particularité de cette commande est qu'elle est sensible à son emplacement dans le fichier source. C'est très facile à expliquer avec quelques exemples, mais beaucoup plus difficile si nous voulons spécifier exactement comment la commande fonctionne et quelles rubriques sont incluses dans la Table des matières dans chaque cas. Commençons donc par les exemples :

- `\placecontent` placé au début du document, avant la première commande de section (partie, chapitre ou section, selon la situation) générera une table des matières complète.

Je ne suis pas vraiment sûr que la table des matières générée par défaut soit complète, je crois qu'elle comprend suffisamment de niveaux de section pour être complète dans la plupart des cas ; mais je soupçonne qu'elle ne dépassera pas le huitième niveau de section. Quoi qu'il en soit, comme mentionné ci-dessus, nous pouvons ajuster le niveau de sectionnement que la TdM atteint avec

```
\setupcombinedlist [content] [list={chapitre, section, sous-section, ...}]
```

- En revanche, cette même commande située à l'intérieur d'une partie, d'un chapitre ou d'une section générera une TdM strictement limité au contenu de l'élément de section concerné, ou en d'autres termes des chapitres, des sections et d'autres niveaux inférieurs de découpage d'une partie spécifique, ou des sections (et d'autres niveaux) d'un chapitre spécifique, ou des sous-sections d'une section spécifique.

En ce qui concerne l'explication technique et détaillée, afin de bien comprendre le fonctionnement par défaut de `\placecontent`, il est essentiel de se rappeler que les différentes sections sont, en fait, des *environnements* pour ConTeXt Mark IV qui commencent par `\start\em TypeSection` et se terminent par `\stop\em TypeSection` et peuvent être contenues dans d'autres commandes de section de niveau inférieur. Donc, en tenant compte de cela, nous pouvons dire que `\placecontent` génère par défaut une table des matières qui comprendra uniquement :

- les éléments qui appartiennent à l'*environnement* (niveau de section) où la commande est placée. Cela signifie que la commande, lorsqu'elle est placée dans un chapitre, ne fera pas apparaître les sections et sous-sections des autres chapitres.
- les éléments qui ont un niveau de section inférieur au niveau correspondant au point où se trouve la commande. Cela signifie que si la commande se trouve dans un chapitre, seules les sections, les sous-sections et les autres niveaux inférieurs sont inclus ; Si la commande se trouve au niveau d'une section, seules les sous-sections, les sous-sous-sections et les autres niveaux inférieurs sont inclus.

En outre, pour que la table des matières soit générée, il faut que `\placecontent` se trouve *avant* la première section du chapitre dans lequel il se trouve, ou avant la première sous-section de la section dans laquelle il se trouve, etc.

Je ne suis pas sûr d'avoir été clair dans l'explication ci-dessus. Peut-être qu'avec un exemple un peu plus détaillé que les précédents, nous pourrons mieux comprendre ce que je veux dire : imaginons la structure suivante d'un document :

- Chapter 1
 - Section 1.1
 - Section 1.2
 - * Subsection 1.2.1
 - * Subsection 1.2.2
 - * Subsection 1.2.3
 - Section 1.3
 - Section 1.4
- Chapter 2

Ainsi : `\placecontent` placée avant le chapitre 1 générera une table des matières complète, similaire à celle générée par `\completecontent` mais sans titre. Mais si la commande est placée à l'intérieur du chapitre 1 et avant la section 1.1, la table des matières sera uniquement celle du chapitre ; et si elle est placée au début de la section 1.2, la table des matières sera uniquement le contenu de cette section. Mais si la commande est placée, par exemple, entre les sections 1.1 et 1.2, elle sera ignorée. Elle sera également ignorée si elle est placée à la fin d'une section, ou à la fin du document.

```

\setuphead[chapter][page=no]

\startchapter[title=chapter 1]

\startcolor[darkgreen]
\placecontent % <=====
\stopcolor

\startsection[title=Section 1.1]
\stopsection
\startsection[title=Section 1.2]

\startcolor[darkred]
\placecontent % <=====
\stopcolor

\startsubsection[title=Sous-section 1.2.1]
\stopsubsection
\startsubsection[title=Sous-section 1.2.2]
\stopsubsection
\startsubsection[title=Sous-section 1.2.3]
\stopsubsection

\stopsection
\startsection[title=Section 1.3]
\stopsection
\startsection[title=Section 1.4]
\stopsection

\stopchapter

\startchapter[title=chapter 2]
\stopchapter

```

1 chapter 1

1.1	Section 1.1	0
1.2	Section 1.2	0
1.2.1	Sous-section 1.2.1	0
1.2.2	Sous-section 1.2.2	0
1.2.3	Sous-section 1.2.3	0
1.3	Section 1.3	0
1.4	Section 1.4	0

1.1 Section 1.1

1.2 Section 1.2

1.2.1	Sous-section 1.2.1	0
1.2.2	Sous-section 1.2.2	0
1.2.3	Sous-section 1.2.3	0

1.2.1 Sous-section 1.2.1

1.2.2 Sous-section 1.2.2

1.2.3 Sous-section 1.2.3

1.3 Section 1.3

1.4 Section 1.4

2 chapter 2

Tout ceci, bien sûr, ne concerne que le cas où la commande ne comporte pas d'options. En particulier, l'option `criterium` modifiera ce comportement par défaut.

Parmi les options autorisées par `\placecontent` je n'en expliquerai que deux, les plus importantes pour la mise en place de la table des matières, et, de plus, les seules qui sont (partiellement) documentées dans le manuel de référence ConTeXt. L'option `criterium`, qui affecte le contenu de la table des matières par rapport à l'endroit du fichier source où se trouve la commande, et l'option `alternative`, qui affecte la présentation générale de la table des matières à générer.

8.1.4 Option de sélection des éléments intégrés à la TdM : l'option criterium

Le fonctionnement par défaut de `\placecontent` vis-à-vis de la position de la commande dans le fichier source a été expliqué ci-dessus. En plus de la commande `\setuptupcombinedlist` vue à la section A qui permet de sélectionner les éléments à intégrer dans la TdM, l'option `criterium` permet d'adapter le fonctionnement. Entre autres, elle peut prendre les valeurs suivantes :

- item `all` : la TdM sera complète, quel que soit l'endroit du fichier source où se trouve la commande.
- `previous` : la table des matières n'inclura que les commandes de section (du niveau auquel nous nous trouvons) précédent la commande `\placecontent`. Cette option est destinée aux TdMs qui sont positionnées à en fin de document ou en fin de la section courante.
- `part`, `chapter`, `section`, `subsection`... : implique que la TdM doit être limitée au niveau de section indiqué.
- `component` : dans les projets multifichiers (voir section 4.6), cette option génère uniquement la TdM correspondant au *composant* (component) où se trouve la commande `\placecontent` ou `\completecontent`.

8.1.5 Mise en page de la TdM : l'option alternative

L'option `alternative` contrôle la présentation générale de la table des matières. Ses principales valeurs peuvent être consultées dans le tableau 8.1.

alternative	Entrées sélectionnées	Remarques
a	Numéro – Titre – Page	Une ligne par entrée
b	Numéro – Titre – Espaces – Page	Une ligne par entrée
c	Numéro – Titre – Ligne de points – Page	Une ligne par entrée
d	Numéro – Titre – Page	TdM continue, entrées bout-à-bout
e	Titre	Encadré
f	Titre	Justification à gauche
g	Titre	Justification à droite ou centrée
		Justification centrée

Tableau 8.1 Ways of formatting the table of contents

Les quatre premières alternatives (a, b, c, d) fournissent toutes les informations de chaque section (son numéro, son titre et le numéro de page où elle commence), et conviennent donc aussi bien aux documents papier qu'aux documents électroniques. Les trois dernières alternatives (e, f, g) ne nous informent que sur le titre, elles ne conviennent donc qu'aux documents électroniques où il n'est pas nécessaire de connaître le numéro de page où commence une section, à condition que la table des matières contienne un lien vers celle-ci, ce qui est le cas par défaut avec ConTeXt.

Par ailleurs, je pense que pour apprécier réellement les différences entre les différentes alternatives, il est préférable que le lecteur génère un document test où il pourra les analyser en détail. Voici quelques illustrations :

```

\setuphead[chapter][page=no]

\startcolor[darkgray]
\placecontent[alternative=a]      % <=====
\stopcolor

\startcolor[darkred]
\placecontent[alternative=b]      % <=====
\stopcolor

\startcolor[darkgreen]
\placecontent[alternative=c]      % <=====
\stopcolor

\startcolor[darkblue]
\placecontent[alternative=d]      % <=====
\stopcolor

\startcolor[darkcyan]
\placecontent[alternative=e]      % <=====
\stopcolor

\startcolor[darkmagenta]
\placecontent[alternative=f]      % <=====
\stopcolor

\startcolor[darkyellow]
\placecontent[alternative=g]      % <=====

\startchapter[title=Chapitre 1]
\startsection[title=Section 1.1]
\startsubsection[title=Sous-section 1.1.1]
\stopsubsection
\stopsection
\stopchapter

```

1	Chapitre 1	0
1.1	Section 1.1	0
1.1.1	Sous-section 1.1.1	0
1	Chapitre 1	0
1.1	Section 1.1	0
1.1.1	Sous-section 1.1.1	0
1	Chapitre 1 0
1.1	Section 1.1 0
1.1.1	Sous-section 1.1.1 0
1	Chapitre 1	0
1.1	Section 1.1	0
1.1.1	Sous-section 1.1.1	0

Chapitre 1
Section 1.1
Sous-section 1.1.1
Chapitre 1
Section 1.1
Sous-section 1.1.1
Chapitre 1
Section 1.1
Sous-section 1.1.1

1 Chapitre 1

1.1 Section 1.1

1.1.1 Sous-section 1.1.1

8.1.6 Format des entrées de la TdM

Nous avons vu que l'option `alternative` de `\placecontent` ou `\completecontent` nous permet de contrôler la *mise en page* générale de la table des matières, c'est-à-dire les informations qui seront affichées pour chaque rubrique, et la présence ou non de sauts de ligne séparant les différentes rubriques. Les ajustements finaux de chaque entrée de la table des matières sont effectués avec la commande `\setuplist` dont la syntaxe est la suivante :

```
\setuplist[Élement][Configuration]
```

où `Élement` fait référence à un type particulier de section. Il peut s'agir de `part`, `chapter`, `section`, etc. On peut également configurer plusieurs éléments en même temps, en les séparant par des virgules. `Configuration` offre jusqu'à 54 possibilités, dont beaucoup, comme d'habitude, ne sont pas expressément documentées ; mais

cela n'empêche pas celles qui sont documentées, ou celles qui ne sont pas suffisamment claires, de permettre un ajustement très complet de la TdM. Je vais maintenant expliquer les options les plus importantes, en les regroupant selon leur utilité, mais avant de les aborder, rappelons qu'une entrée de la TdM, selon la valeur de l'option `alternative`, peut comporter jusqu'à trois éléments différents : Le numéro de section, le titre de la section et le numéro de page. Les options de configuration nous permettent de configurer les différents composants de manière globale ou séparée :

- **Inclusion ou exclusion des différents composants** : Si nous avons choisi une alternative qui inclut, en plus du titre, le numéro de section et le numéro de page (alternatives « a » « b » « c » ou « d »), les options `headnumber=no` ou `pagenumber=no` indiquent pour le niveau spécifique que nous configurons, que le numéro de section (`headnumber`) ou le numéro de page (`pagenumber`) ne doivent pas être affichés.
- **Couleur et style** : Nous savons déjà que l'entrée qui génère une section spécifique dans la TdM peut avoir (selon l'alternative) jusqu'à trois composants différents : le numéro de section, le titre et le numéro de page. Nous pouvons indiquer conjointement le style et la couleur des trois composants à l'aide des options `style` et `couleur`, ou le faire individuellement pour chaque composant au moyen des options `numberstyle`, `textstyle` et `pagestyle` (pour le style) et `numbercolor`, `textcolor` ou `pagecolor` pour la couleur.

Pour contrôler l'apparence de chaque entrée, en plus du style lui-même, nous pouvons appliquer une commande à l'entrée entière ou à l'un de ses différents éléments. Pour cela, il existe les options `command`, `numbercommand`, `pagecommand` et `textcommand`. La commande indiquée ici peut être une commande standard ConTeXt ou une commande de notre propre création. Le numéro de section, le texte du titre et le numéro de page seront passés comme arguments à l'option `command`, tandis que le titre de la section sera passé comme argument à `textcommand` et le numéro de page à `pagecommand`. Ainsi, par exemple, la phrase suivante fera en sorte que les titres de section soient écrits en (fausses) petites majuscules :

```
\setuphead[chapter][page=no]

\setuplist
  [section]
  [textcommand=\cap,
  pagecolor=darkgreen,
  numberstyle=bold]

\placecontent[alternative=c]      % <=====

\startchapter[title=Chapitre 1]
\startsection[title=Section 1.1]
\startsubsection[title=Sous-section 1.1.1]
\stopsubsection
\stopsection
\stopchapter
```

1	Chapitre 1	0
1.1	SECTION 1.1	0
1.1.1	Sous-section 1.1.1	0

1 Chapitre 1

1.1 Section 1.1

1.1.1 Sous-section 1.1.1

- **Séparation des autres éléments de la TdM** : Les options `before` et `after` nous permettent d'indiquer les commandes qui seront exécutées avant (`before`) et après (`after`) la composition de l'entrée de la table des matières. Normalement, ces commandes sont utilisées pour définir l'espacement ou un élément de séparation (tel un trait horizontal) entre les entrées précédentes et suivantes.
 - **Indentation d'un élément** : défini avec l'option `margin` qui nous permet de définir l'espace d'indentation gauche des entrées du niveau que nous configurons.
 - **Hyperliens intégrés à la TdM** : Par défaut, les entrées de l'index comprennent un lien vers la page du document où commence la section en question. L'option `interaction` permet de désactiver cette fonction (`interaction=no`) ou de limiter la partie de l'entrée d'index où se trouvera l'hyperlien, qui peut être le numéro de section (`interaction=number` ou `interaction=section-number`), le titre de la section (`interaction=text` ou `interaction=title`) ou le numéro de page (`interaction=page` ou `interaction=pagenumber`).
 - *Autres aspects* :
 - `width` et `distance` : spécifient respectivement la largeur accordé à l'affichage du numéro et la distance de séparation entre l'espace d'affichage du numéro et le titre de la section. Il peut s'agir d'une dimension (ou du mot clé `fit` pour `width` qui définit la largeur exacte du numéro de section).
 - `numberalign` : indique l'alignement des éléments de numérotation ; il peut être `left`, `right`, `middle`, `flushright`, `flushleft`, `inner`, `outer`
- Voici un exemple qui combine ces trois dernières options :

```
\setuphead[chapter][page=no]

\setuplist[chapter] [width=15mm, margin=0mm,
                   distance=3mm,
                   numberalign=flushright]
\setuplist[section] [width=15mm, margin=0mm,
                     distance=6mm,
                     numberalign=flushright]
\setuplist[subsubsection] [width=15mm, margin=0mm,
                          distance=9mm,
                          numberalign=flushright]

\placecontent

\startchapter[title=Chapitre 1]
\startsection[title=Section 1.1]
\startsubsection[title=Sous-section 1.1.1]
\stopsubsection
\stopsection
\stopchapter
```

1	Chapitre 1	0
1.1	Section 1.1	0
1.1.1	Sous-section 1.1.1	0

1 Chapitre 1

1.1 Section 1.1

1.1.1 Sous-section 1.1.1

- `symbol` : permet de remplacer le numéro de section par un *symbole*. Trois valeurs possibles sont prises en charge : `one`, `two` et `three`. La valeur `none` de cette option supprime le numéro de section de la table des matières.

Parmi les multiples options de configuration de la TdM, aucune ne nous permet de contrôler directement l'espacement entre les lignes. Celui-ci sera, par défaut, celui qui s'applique à l'ensemble du document. Souvent, cependant, il est préférable que les lignes de la table des matières soient légèrement plus serrées que le reste du document. Pour ce faire, nous devons inclure la commande qui génère la table des matières (`\placecontent` ou `\completecontent`) dans un groupe où l'espacement interligne est défini différemment. Par exemple :

```
\setuphead[chapter][page=no]
\start
  \startcolor[darkred]
  \setupinterlinespace[6mm]
  \placecontent
  \stopcolor
\stop

\start
  \startcolor[darkyellow]
  \setupinterlinespace[small]
  \placecontent
  \stopcolor
\stop

\startchapter[title=Chapitre 1]
\startsection[title=Section 1.1]
  \startsubsection[title=Sous-section 1.1.1]
  \stopsubsection
\stopsection
\stopchapter
```

1	Chapitre 1	0
1.1	Section 1.1	0
1.1.1	Sous-section 1.1.1	0
1	Chapitre 1	0
1.1	Section 1.1	0
1.1.1	Sous-section 1.1.1	0

1 Chapitre 1

1.1 Section 1.1

1.1.1 Sous-section 1.1.1

8.1.7 Ajustements manuels

Nous avons déjà expliqué les deux commandes fondamentales pour générer des tables des matières (`\placecontent` et `\completecontent`), ainsi que leurs options. Ces deux commandes permettent de générer automatiquement des tables des matières, construites à partir des sections numérotées existantes dans le document, ou dans le bloc ou le segment du document auquel la table des matières fait référence. Je vais maintenant expliquer certaines configurations que nous pouvons effectuer pour que le contenu de la table des matières ne soit pas aussi *automatique* et plus *personnalisé*. Cela implique :

- la possibilité d'inclure également certains titres de sections non numérotées dans la table des matières.
- la possibilité d'intégrer dans la table des matières une entrée particulière qui ne correspond pas à une section numérotée.

- la possibilité d'exclure une section numérotée particulière de la table des matières.
- la possibilité que le titre d'une section particulière figurant dans la table des matières ne coïncide pas exactement avec le titre figurant dans le corps du document.

A. Inclure les sections non numérotées dans la TdM

Le mécanisme par lequel ConTeXt construit la TdM implique que toutes les sections numérotées sont automatiquement incluses, ce qui, comme je l'ai déjà dit (voir section 7.4.2) dépend des deux options (`number` et `incrementnumber`) que nous pouvons modifier avec `\setuphead` pour chaque type de section. Il a également été expliqué qu'un type de section où `incrementnumber=yes` et `number=no` serait une section numérotée *en interne* mais pas *en externe* (c'est à dire pas de façon visible).

Par conséquent, si nous voulons qu'un type de section non numéroté particulier – par exemple, `subject` – soit inclus dans la table des matières, nous devons modifier la valeur de l'option `incrementnumber` pour ce type de section, en lui attribuant la valeur `yes`, puis inclure ce type de section parmi ceux qui doivent être affichés dans la table des matières, ce qui se fait, comme expliqué précédemment, avec `\setupcombinedlist`.

Nous pouvons ensuite, si nous le souhaitons, formater cette entrée en utilisant `\setuplist` de la même manière que les autres ; par exemple :

```
\setuphead [chapter] [page=no]

\setuphead [subject]
    [incrementnumber=yes]
\setuplist [subject] [color=darkred]

\setupcombinedlist
    [content]
    [list={chapter, subject, section, subsection}]

\placecontent

\startchapter[title=Chapitre 1]
\startsection[title=Section 1.1]
\stopsection
\startsubject[title=Sujet 1.2]
\stopsubject
\stopchapter
```

1	Chapitre 1	0
1.1	Section 1.1	0
1.2	Sujet 1.2	0

1 Chapitre 1

1.1 Section 1.1

1.2 Sujet 1.2

Note : La procédure qui vient d'être expliquée inclura toutes les occurrences dans notre document du type de section non numérotée concerné (dans notre exemple, les sections de type `subject`). Si l'on ne souhaite inclure qu'une occurrence particulière de ce type de section dans la table des matières, il est préférable de le faire par la procédure expliquée ci-dessous.

B. Ajouter des entrées manuellement à la TdM

On peut envoyer soit une entrée (simulant l'existence d'une section qui n'existe pas réellement), soit une commande à la table des matières, à partir de n'importe quel point du fichier source.

Pour envoyer une entrée qui simule l'existence d'une section qui n'existe pas réellement, utilisez la fonction `\writetolist` dont la syntaxe est :

```
\writetolist [Type de Section] [Options] {Numéro} {Texte}
```

dans laquelle :

- Le premier argument indique le niveau que doit avoir cette entrée de section dans la table des matières : `chapter`, `section`, `sous-section`, etc.
- Le deuxième argument, qui est facultatif, permet de configurer cette entrée d'une manière particulière. Si l'entrée envoyée manuellement est omise, elle sera formatée comme le sont toutes les entrées du niveau indiqué par le premier argument ; je dois cependant signaler que dans mes tests, je n'ai pas réussi à le faire fonctionner.

Tant dans la liste officielle des commandes de ConTeXt (voir [section 3.6](#)) que dans le wiki, on nous dit que cet argument permet les mêmes valeurs que `\setuplist` qui est la commande qui nous permet de formater les différentes entrées de la COT. Mais, j'insiste, dans mes tests je n'ai pas réussi à modifier de quelque manière que ce soit l'apparence de l'entrée de la TdM envoyée manuellement.

- Le troisième argument est censé refléter la numérotation que possède l'élément envoyé à la COT, mais je n'ai pas non plus réussi à le faire fonctionner dans mes tests.
- Le dernier argument comprend le texte à envoyer à la table des matières.

Ceci est utile, par exemple, si nous voulons envoyer une section non numérotée particulière, mais seulement celle-ci à la table des matières. Dans [section A](#), il est expliqué comment obtenir qu'une catégorie entière de sections non numérotées soit envoyée à la table des matières ; mais si nous voulons seulement y envoyer une occurrence particulière d'un type de section, il est plus pratique d'utiliser la commande `\writetolist` localement. Ainsi, par exemple, si nous voulons que la section de notre document contenant la bibliographie ne soit pas une section numérotée, mais qu'elle soit tout de même incluse dans la table des matières, nous devons écrire :



```
\setuphead [chapter] [page=no]

\placecontent

\startchapter[title=Chapitre 1]
\startsection[title=Section 1.1]
\stopsection
\startsubject[title=Sujet]
\writetolist[section]{53}{Sujet important}
\stopsubject
\stopchapter
```

1	Chapitre 1	0
1.1	Section 1.1	0
	Sujet important	0

1 Chapitre 1

1.1 Section 1.1

Sujet

Voyez comment nous utilisons la version non numérotée de `section`, qui est `subject`, pour la section mais nous l'envoyons à l'index, manuellement, comme s'il s'agissait d'une section numérotée (`section`).

Une autre commande destinée à influencer manuellement la table des matières est `\writebetweenlist` qui est utilisée pour envoyer non pas une entrée elle-même, mais une *commande* à la table des matières, depuis un point particulier du document. Par exemple, si nous voulons inclure une ligne entre deux éléments de la TdM, nous pouvons écrire ce qui suit à n'importe quel endroit du document situé entre les deux sections concernées :

```
\setuphead [chapter] [page=no]

\placecontent

\startchapter[title=Chapitre 1]
\startsection[title=Section 1.1]
\stopsection
\writebetweenlist
[section]
[location=here]
{\hrule}
\startsection[title=Section 1.2]
\stopsection
\stopchapter
```

1	Chapitre 1	0
1.1	Section 1.1	0
1.2	Section 1.2	0

1 Chapitre 1

1.1 Section 1.1

1.2 Section 1.2

C. Exclure de la TdM une section particulière appartenant à un type de section qui est inclus dans le TdM

La table des matières est construite à partir de *types de section* établis, comme nous le savons déjà, par l'option `list` de `\setupcombinedlist`. Par conséquent, si un certain *type de section* doit apparaître dans la table des matières, il n'y a aucun moyen d'en exclure une section particulière que nous ne voulons pas voir figurer dans la table des matières, pour quelque raison que ce soit.

Normalement, si l'on ne veut pas qu'une section apparaisse dans la table des matières, il faut utiliser son équivalent non numéroté, c'est-à-dire, par exemple, `title` au lieu de `chapter`, `subject` au lieu de `section`, etc. Ces sections ne sont pas envoyées à la TdM, et ne sont pas non plus numérotées.

Cependant, si pour une raison quelconque, nous voulons qu'une certaine section soit numérotée mais n'apparaisse pas dans la table des matières, même si d'autres types de ce genre le font, nous pouvons utiliser une *astuce* qui consiste à créer un nouveau type de section qui est un clone de la section en question. Par exemple :

```
\definehead[MySubsection][subsection]
\placecontent
\startsection [title={Première section}]
\stopsection
\startsubsection [title={Première subsection}]
\stopsubsection
\startMySubsection [title={Seconde subsection}]
\stopMySubsection
\startsubsection [title={Troisième subsection}]
\stopsubsection
```

1	Première section	0
1.1	Première subsection	0
1.3	Troisième subsection	0
1	Première section	
1.1	Première subsection	
1.2	Seconde subsection	
1.3	Troisième subsection	

Ainsi, lors de l'insertion d'une section de type `MySubsection`, le compteur de sous-sections augmentera, puisque cette section est un *clone* des sous-sections, mais la TdM ne sera pas modifiée, puisque par défaut elle n'inclut pas les sections de types `MySubsection`.

D. Textes du titre différents entre TdM et corps du document

Si nous voulons un texte de titre d'une section particulière différent entre celui inclus dans la table des matières et celui affiché dans le corps du document, nous devons créer la section non pas avec la syntaxe traditionnelle (`SectionType{Titre}`) mais avec `\SectionType [Options]`, ou avec `\startSectionType [Options]`, et attribuer le texte que l'on souhaite voir écrit dans la Table des matières à l'option `list` (voir section ??).

```
\placecontent
\startsection
  [title={Une introduction approximative
et légèrement répétitive à la réalité de
l'évidence},
  list={Une introduction à la réalité de
l'évidence}]
\stopsection
```

1	Une introduction à la réalité de l'évidence	0
1	Une introduction approximative et légèrement répétitive à la réalité de l'évidence	

8.2 Listes, listes combinées et tables des matières basées sur une liste

En interne, pour ConTeXt une table des matières n'est rien de plus qu'une *liste combinée*, qui, à son tour, comme son nom l'indique, consiste en une combinaison de listes simples. Par conséquent, la notion de base à partir de laquelle ConTeXt construit la table des matières est celle d'une liste. Plusieurs listes sont combinées pour former une table des matières. Par défaut, ConTeXt contient une liste combinée prédéfinie appelée « `content` » et c'est avec elle que les commandes examinées jusqu'à présent fonctionnent : `\placecontent` et `\completecontent`.

8.2.1 Listes de ConTeXt

Dans ConTeXt, une *liste* est plus précisément une liste d'éléments numérotés à propos desquels nous devons nous souhaitons conservers trois informations :

1. leur numéro.
2. leur texte, qui peut aussi être leur nom, leur titre.
3. leur page d'apparition.

Cela est utilisé avec les sections numérotées, mais aussi avec d'autres éléments du document tels que les images, les tableaux, etc. En général, les éléments pour lesquels il existe une commande dont le nom commence par `\place` qui les positionne dans le document comme `\placetable`, `\placefigure`, etc.

Dans tous ces cas, ConTeXt génère automatiquement une liste des différentes occurrences du type d'élément en question, avec pour chaque occurrence : son numéro, son titre (ou texte) et sa page. Ainsi, par exemple, il existe une liste de chapitres, appelée `chapter`, une autre de sections, appelée `section` ; mais aussi une autre de tableaux (appelée `table`) ou d'images (appelée `figure`). Les listes générées automatiquement par ConTeXt sont toujours appelées de la même manière que l'élément qu'elles stockent.

Une liste sera également générée automatiquement si nous créons, par exemple, un nouveau type de section numérotée : lorsque nous la créons, nous créons aussi implicitement la liste qui les stocke. Et si, pour une section non numérotée par défaut, nous activons l'option `incrementnumber=yes`, ce qui en fait une section numérotée, nous créerons aussi implicitement une liste portant ce nom.

Outre les listes implicites (définies automatiquement par ConTeXt), nous pouvons créer nos propres listes avec `\definelist`, dont la syntaxe est la suivante

```
\definelist [ListName] [Configuration]
```

Des éléments sont ajoutés de la liste :

- Dans les listes prédéfinies par ConTeXt, ou créées par lui suite à la création d'un nouvel objet flottant (voir [section 13.5](#)), automatiquement chaque fois qu'un élément de la liste est inséré dans le document, soit par une commande de section, soit par la commande `\placeQuoiQueCeSoit` pour les autres types de listes, par exemple : `\placefigure`, insère une image dans le document, mais elle insère également l'entrée correspondante dans la liste.
- Manuellement dans tout type de liste avec `\writetolist [ListName]`, déjà expliqué dans [subsection B de section 8.1.7](#). La commande `\writebetweenlist` est également disponible. Elle a également été expliquée dans cette section.

Une fois qu'une liste a été créée et que tous ses éléments y ont été inclus, les trois commandes de base qui s'y rapportent sont `\setuplist`, `\placelist` et `\completelist`. La première nous permet de configurer l'aspect de la liste ; les deux dernières insèrent la liste en question à l'endroit du document où elle les trouve. La différence entre `\placelist` et `\completelist` est similaire à la différence entre `\placecontent` et `\completecontent` : `complete` introduit la liste dans une section dédiée, avec un titre dédié, de niveau `chapter / title`. (voir les sections [8.1.2](#) et [8.1.3](#)).

Donc, par exemple `\placelist[section]` insère une liste des sections, y compris un lien hypertexte vers celles-ci (si l'interactivité du document est activée et si, dans `\setuplist`, nous n'avons pas défini `interaction=no`). Une liste de sections n'est pas exactement la même chose qu'une table des matières basée sur des sections : l'idée d'une table des matières inclut généralement aussi les niveaux supérieurs et inférieurs (sous-sections, sous-sous-sections, etc.). Mais une liste de sections ne comprendra que les sections elles-mêmes.

```
\placelist[section]

\startsection[title=Section 1.1]
\stopsection

\startsection[title=Section 1.2]
\startsubsection[title=Sous-section 1.2.1]
\stopsubsection
\startsubsection[title=Sous-section 1.2.2]
\stopsubsection
\startsubsection[title=Sous-section 1.2.3]
\stopsubsection
\stopsection

\startsection[title=Section 1.3]
\stopsection
\startsection[title=Section 1.4]

\stopsection
```

1	Section 1.1	0
2	Section 1.2	0
3	Section 1.3	0
4	Section 1.4	0
1	Section 1.1	
2	Section 1.2	
2.1	Sous-section 1.2.1	
2.2	Sous-section 1.2.2	
2.3	Sous-section 1.2.3	
3	Section 1.3	
4	Section 1.4	

La syntaxe de ces commandes est la suivante :

```
\placelist[ListName] [Options]
\setuplist[ListName] [Configuration]
```

Les options de `\setuplist` ont déjà été expliquées dans section 8.1.6, et les options de `\placelist` sont les mêmes que celles de `\placecontent` (voir section 8.1.3).

8.2.2 Listes ou index d'images, de tableaux et d'autres éléments

D'après ce qui a été dit jusqu'à présent, on peut voir que, puisque ConTeXt crée automatiquement une liste d'images placées dans un document avec la commande `\placefigure`, générer une liste ou un index d'images à un point particulier de notre document est aussi simple que d'utiliser la commande `\placelist[figure]`. Et si nous voulons générer une liste avec un titre (similaire à ce que nous obtenons avec `\completecontent`), nous pouvons le faire avec `\completelist[figure]`. Nous pouvons faire de même avec les quatre autres types prédéfinis d'objets flottants dans ConTeXt : tableaux (« `table` »), graphiques (« `graphic` »), *intermezzi* (« `intermezzo` ») et formules chimiques (« `chemical` »). ConTeXt comprend

également des commandes qui les génère dans leur version sans titre (`\placelistoffigures`, `\placelistoftables`, `\placelistofgraphics`, `\placelistofintermezzi` et `\placelistofchemicals`), et d'autre qui les génère avec titre (`\completelistoffigures`, `\completelistoftables`, `\completelistofgraphics`, `\completelistofintermezzi` et `\completelistofchemicals`), de manière similaire à `\completecontent`.

De la même manière, pour les objets flottants que nous avons nous-mêmes créés (voir section 13.5), les commandes `\placelistof<FloatName>` et `\completelistof<FloatName>` seront automatiquement créés.

Pour les listes que nous avons créées avec `\definelist[ListName]`, nous pouvons créer un index avec `\placelist[ListName]` ou avec `\completelist[ListName]`.

8.2.3 Listes combinées

Une liste combinée est, comme son nom l'indique, une liste qui combine des éléments provenant de différentes listes précédemment définies. Par défaut, ConTeXt définit une liste combinée pour les tables de contenu dont le nom est « `content` », mais nous pouvons créer d'autres listes combinées avec `\definecombinedlist` dont la syntaxe est :

```
\definecombinedlist[Nom][Listes][Options]
```

où :

- *Nom* : est le nom que portera la nouvelle liste combinée.
- *Listes* : désigne les noms des listes à combiner, séparés par des virgules.
- *Options* : Options de configuration de la liste. Elles peuvent être indiquées au moment de la définition de la liste, ou, de préférence, lorsque la liste est invoquée. Les principales options (qui ont déjà été expliquées) sont les suivantes : `criterium` (sous-section 8.1.4 de section 8.1.3) et `alternative` (dans sous-section ?? de la même section).

Un effet collatéral de la création d'une liste combinée avec `\definecombinedlist` est qu'elle crée également une commande appelée `\placeListNom` qui sert à invoquer la liste, c'est-à-dire à l'inclure dans le fichier de sortie. Ainsi, par exemple,

```
\definecombinedlist[TdM]
\definecombinedlist[content]
```

créera les commandes `\placeTdM` et `\placecontent`.

Mais attendez, `\placecontent` ! N'est-ce pas la commande qui est utilisée pour créer une table des matières *normal* ? En effet, cela signifie que la table des matières standard est en fait créée par ConTeXt au moyen de la commande suivante :

```
\definecombinedlist
[content]
[part, chapter, section, subsection,
 subsubsection, subsubsubsection,
 subsubsubsubsection]
```

Une fois notre liste combinée définie, nous pouvons la configurer (ou la reconfigurer) avec `\setupcombinedlist` qui permet les options déjà expliquées `criterium` (voir [sous-section 8.1.4](#) dans [section 8.1.3](#)) et `alternative` (voir [subsection 8.1.5](#) dans la même section), ainsi que l'option `list` pour *modifier* les listes incluses dans la liste combinée.

La liste officielle des commandes ConTeXt (voir [section 3.6](#)) ne mentionne pas l'option `list` parmi les options autorisées pour `\setupcombinedlist`, mais elle est utilisée dans plusieurs exemples d'utilisation de cette commande dans le wiki (qui, par ailleurs, ne la mentionne pas non plus dans la page consacrée à cette commande sauf en exemple). J'ai également vérifié que l'option fonctionne.

8.3 Index

8.3.1 Génération de l'index

Générer un index consiste à générer la liste des sujets considérés comme importants ou significatifs et précisant les pages y faisant référence. L'index est généralement située à la fin d'un document.

Lorsque les livres étaient composés à la main, la création d'un index des sujets était une tâche complexe et fastidieuse. Tout changement dans la pagination pouvait affecter toutes les entrées de l'index. Par conséquent, ils n'étaient pas très courants. Aujourd'hui, les mécanismes informatiques de composition font que, même si la tâche est susceptible de rester fastidieuse, elle n'est plus aussi complexe étant donné qu'il n'est pas si difficile pour un système informatique de maintenir à jour la liste des données associées à une entrée d'index.

Pour générer un index par sujet, nous avons besoin de :

1. Déterminer les mots, les termes ou les concepts qui doivent faire partie de l'article. Il s'agit d'une tâche que seul l'auteur peut accomplir.
2. Vérifier à quels endroits du document chaque entrée du futur index apparaît. Bien que, pour être précis, plus que *vérifier* les endroits du fichier source où le concept ou la question est abordé, ce que nous faisons lorsque nous travaillons avec ConTeXt consiste à *marquer* ces endroits, en insérant une commande qui servira ensuite à générer l'index automatiquement. C'est la partie la plus fastidieuse.
3. Enfin, nous générerons et mettons en forme l'index en le plaçant à l'endroit de notre choix dans le document. Cette dernière opération est assez simple avec ConTeXt et ne nécessite qu'une seule commande : `\placeindex` ou `\completeindex`.

A. The prior definition of the entries in the index and the marking of the points in the source file that refer to them

The fundamental work is in the second step. It is true that computer systems also facilitate it in the sense that we can do a global text search to locate the places in the source file where a specific subject is treated. But we should also not blindly rely on such text searches : a good subject index must be able to detect every spot where a particular subject is being discussed, even if this is done without using the *standard* term to refer to it.

To *mark* an actual point in the source file, associating it with a word, term or idea that will appear in the index, we use the `\index` command whose syntax is as follows :

`\index[Alphabetical][Index entry]`

where *Alphabetical* is an optional argument that is used to indicate an alternative text to that of the index entry itself in order to sort it alphabetically, and *Index entry* is the text that will appear in the index, associated with this mark. We can also apply the formatting features that we wish to use, and if reserved characters appear in the text, they must be written in the usual way in ConTEXt.

The possibility of alphabetising an index entry in a way different from how it is actually written, is very useful. Think, for example, of this document, if I want to generate an entry in the index for all references to the \TeX command. For example, the sequence \index{\backslash TeX} will list the command not by the « t » in « TeX », but among the symbols, since the term sent to the index begins with a backslash. This is done by writing \index[tex]{\backslash TeX}.

The *index entries* will be the ones we want. For a subject index to be really useful we have to work a little harder at asking what concepts the reader of a document is most likely to look for ; so, for example, it may be better to define an entry as « disease, Hodgkins » than defining it as « Hodgkin's disease », since the more inclusive term is « disease ».

By convention, entries in a subject index are always written in lower case, unless they are proper names.

If the index has several levels of depth (up to three are allowed) to associate a particular index entry with a specific level the « + » character is used. As follows :

```
\index{Entry 1+Entry 2}
\index{Entry 1+Entry 2+Entry 3}
```

In the first case we defined a second level entry called *Entry 2* that will be a sub-entry of *Entry 1*. In the second case we defined a third level entry called *Entry 3* that will be a sub-entry of *Entry 2*, which in turn is a sub-entry of *Entry 1*. For example

```
My \index{dog}dog, is a \index{dog+greyhound}greyhound called Rocket.
He does not like \index{cat+stray}stray cats.
```

It is worth noting some details of the above :

- The \index command is usually placed *before* the word it is associated with and is normally not separated from it by a blank space. This is to ensure that the command is on the exact same page as the word it is linked to :
 - If there were a space separating them, there could be the possibility that ConTEXt would choose just that space for a line break which could also end up being a page break, in which case the command would be on one page and the word it is associated with on the next page.
 - If the command were to come *after* the word, it would be possible for this word to be broken by syllables and a line break inserted between two of its syllables that would also be a page break, in which case the command would be pointing to the next page beginning with the word it points to.
- See how second level terms are introduced in the second and third appearances of the command.

- Also check how, in the third use of the `\index` command, although the word that appears in the text is « cats », the term that will be sent to the index is « cat ».
- Finally : see how three entries for the subject index have been written in just two lines. I said before that marking the precise places in the source file is tedious. I will now add that marking too many of them is counter-productive. Too extensive an index is by no means preferable to a more concise one in which all the information is relevant. That is why I said before that deciding which words will generate entry in the index should be the result of a conscious decision by the author.

If we want our index to be truly useful, terms that are used as synonyms must be grouped in the index under one head term. But since it is possible for the reader to search the index for information by any of the other head terms, it is common for the index to contain entries that refer to other entries. For example, the subject index of a civil law manual could just as easily be something like

contractual invalidity
see *nullity*.

We achieve this not with the `\index` command but with `\seeindex` whose format is :

```
\seeindex[Alphabetical] {Entry1} {Entry2}
```

where *Entry1* is the index entry that will refer to the other ; and *Entry2* is the reference target. In our previous example we would have to write :

```
\seeindex{contractual invalidity}{nullity}
```

In `\seeindex` we can also use the « + » sign to indicate sub-levels

for either of its two arguments in square brackets.

B. Generating the final index

Once we have marked all the entries for the index in our source file, the actual generation of the index is carried out using the `\placeindex` or `\completindex` commands. These two commands scan the source file for the `\index` commands, and generate a list of all the entries that the index should have, associating a term with the page number corresponding to where it found the `\index` command. Then they alphabetically order the list of terms that appear in the index and merge cases where the same term appears more than once, and finally, they insert the correctly formatted result in the final document.

The difference between `\placeindex` and `\completindex` is similar to the difference between `\content` and `\completecontent` (see section 8.1.2) : `\placeindex` is limited to generating the index and inserting it, while `\completindex` previously inserts a new chapter in the final document, called « Index » by default, inside which the index will be typeset.

8.3.2 Formatage de l'index

Les index de sujets sont une application particulière d'une structure plus générale que ConTeXt appelle « *register* » ; par conséquent l'index est formaté avec la commande :

```
\setupregister [index] [Configuration]
```

Avec cette commande, nous pouvons :

- Déterminer à quoi ressemblera l'index avec ses différents éléments. A savoir :
 - Les titres de l'index qui sont généralement des lettres de l'alphabet. Par défaut, ils sont en minuscules. Avec `alternative=A`, nous pouvons les mettre en majuscules.
 - Les entrées elles-mêmes, et leur numéro de page. L'apparence dépend des options `textstyle`, `textcolor`, `textcommand` et `deeptextcommand` pour l'entrée elle-même, et `pagestyle`, `pagecolor` et `pagecommand`, pour le numéro de page. Avec `pagenumber=no`, on peut aussi générer un index des sujets sans numéro de page (mais je ne sais pas si cela peut être utile).
 - L'option `distance` spécifie la largeur de séparation entre le nom d'une entrée et les numéros de page ; mais elle mesure aussi la quantité d'indentation pour les sous-entrées.
- Les noms des options `style`, `textstyle`, `pagestyle`, `color`, `textcolor`, et `pagecolor` sont suffisamment clairs pour nous dire ce que chacune fait, je pense. Pour les options `command`, `pagecommand`, `textcommand` et `deeptextcommand`, je me réfère à l'explication des options de même nom dans [section 7.4.4](#), concernant la configuration des commandes de section.
- Pour définir l'apparence générale de l'index, ce qui inclut, entre autres, les commandes à exécuter avant (`before`) ou après (`after`) l'index, le nombre de colonnes qu'il doit avoir (`n`), si les colonnes doivent être d'égale hauteur ou non (`balance`), l'alignement des entrées (`align`), etc.

8.3.3 Crédit d'autres index

J'ai expliqué l'index des sujets comme si un seul index de ce type était possible dans un document ; mais la vérité est que les documents peuvent avoir autant d'index que souhaité. Il peut y avoir un index des noms de personnes, par exemple, qui rassemble les noms des personnes mentionnées dans le document, avec une indication de la page où elles sont citées. Il s'agit toujours d'une sorte d'index. Dans un texte juridique, nous pourrions également créer un index spécial pour les mentions du Code civil ; ou, dans un document comme le présent, un index des macros expliquées dans celui-ci, etc.

Pour créer un index supplémentaire dans notre document, nous utilisons la commande `\defineregister` dont la syntaxe est :

```
\defineregister [NomIndex] [Configuration]
```

où *NomIndex* est le nom que portera le nouvel index, et *Configuration* contrôle son fonctionnement. Il est également possible de configurer l'index ultérieurement au moyen de la fonction

```
\setupregister [NomIndex] [Configuration]
```

Une fois qu'un nouvel index nommé *NomIndex* a été créé, nous disposons de la commande `\NomIndex` pour marquer les entrées de cet index de la même manière que les entrées sont marquées avec `\index`. La commande `\seeNomIndex` nous permet également de créer des entrées qui font référence à d'autres entrées.

Par exemple, nous pouvons créer un index des commandes ConTeXt dans ce document avec la commande :

```
\defineregister[MaMacro]  
  
La commande \tex{etbim} permet de...  
\MaMacro{etbim}  
  
\startsubject[title=Index des macros]  
\placeMaMacro  
\stopsubject
```

La commande `\etbim` permet de...

Index des macros

e
etbim 0

qui crée la commande `\MaMacro`. Cela me permet de marquer toutes les références aux commandes ConTeXt comme une entrée d'index, puis de générer l'index avec `\placeMaMacro` ou `\completeMaMacro`.

La création d'un nouvel index active la commande `\NomIndex` pour le marquer comme entrée, et les commandes `\placeNomIndex` et `\completeNomIndex` pour générer l'index. Mais ces deux dernières commandes sont en fait des abréviations de deux commandes plus générales appliquées à l'index en question. Ainsi, `\placeNomIndex` est équivalent à `\placeregister[NomIndex]` et `\completeNomIndex` est équivalent à `\completeregister[NomIndex]`.

Chapitre 9

Références et hyperliens

Table of Contents : 9.1 Types de référence ; 9.2 Références internes ; 9.2.1 L'étiquette de la cible ; 9.2.2 Commandes au point de référence d'origine pour récupérer les données du point cible ; A Commandes de base pour récupérer les informations d'une étiquette ; B Récupération des informations associées à une étiquette avec la commande `\ref` ; C Déceler où le lien mène ; 9.2.3 Génération automatique de préfixes pour éviter les étiquettes en double ; 9.3 Documents électroniques interactifs ; 9.3.1 Permettre l'interactivité dans les documents ; 9.3.2 Configuration de base pour les éléments interactifs ; 9.4 Hyperliens vers des documents externes ; 9.4.1 Commandes de définition d'hyperlien (sans les introduire dans le document) ; 9.4.2 Commandes d'insertion d'hyperlien dans le document ; 9.5 Création de signets dans le PDF final ; 9.6 Pièces jointes ; 9.7 Références bibliographiques ;

9.1 Types de référence

Les documents scientifiques et techniques abondent de références :

- Parfois, ils se réfèrent à d'autres documents qui sont à la base de ce qui est expliqué, ou qui contredisent ce qui est expliqué, ou qui développent ou nuancent l'idée traitée, etc. Dans ce cas, la référence est dite *externe* et, si l'on veut que le document soit rigoureux sur le plan académique, la référence prend la forme de *citations* de la littérature.
- Mais il est également fréquent qu'un document, dans l'une de ses sections, fasse référence à une autre de ses sections, auquel cas la référence est dite *interne*. Lorsqu'un point du document commente un aspect particulier d'une image, d'un tableau, d'une note ou d'un élément de même nature, en s'y référant par son numéro ou par la page où il se trouve, il s'agit également d'une référence *interne*.

Pour des raisons de précision, les références internes doivent viser un endroit précis et facilement identifiable dans le document. C'est pourquoi ce type de références renvoie toujours soit à des éléments numérotés (comme, par exemple, lorsque nous disons « voir tableau 3.2 », ou « chapitre 7 »), soit à des numéros de page. Les références vagues du type « comme nous l'avons déjà dit » ou « comme nous le verrons plus loin » ne sont pas de véritables références, et il n'y a pas d'exigence particulière pour les mettre en forme, ni d'outil spécial pour le faire. De plus, je dissuade personnellement mes étudiants de doctorat ou de maîtrise d'utiliser cette pratique de manière habituelle.

Les références internes sont aussi communément appelées « références croisées » bien que dans ce document j'utiliserai simplement le terme « références » en général, et « références internes » lorsque je souhaite être spécifique.

Afin de clarifier la terminologie que j'utilise pour les références, j'appelleraï le point du document où une référence est introduite, l'« origine », et l'endroit vers lequel elle pointe, la quotationnable. De cette façon, nous dirons qu'une référence est interne lorsque l'origine et la cible sont dans le même document, et externe lorsque l'origine et la cible sont dans des documents différents.

Du point de vue de la composition du document :

- Les références externes ne posent aucun problème particulier et ne nécessitent donc, en principe, aucun outil pour les introduire : toutes les données dont j'ai besoin à partir du document cible sont à ma disposition et je peux les utiliser dans la référence. Toutefois, si le document d'origine est un document électronique et que le document cible est également disponible sur le Web, il est possible d'inclure dans la référence un lien hypertexte permettant de passer directement à la cible par un clic. Dans ce cas, on peut dire que le document d'origine est *interactif*.
- En revanche, les références internes posent un défi pour la composition du document. En effet, quiconque a l'expérience de la préparation de documents scientifiques et techniques moyennement longs sait qu'il est presque inévitable que la numérotation des pages, des sections, des images, des tableaux, des théorèmes ou d'autres éléments similaires à ce qui est indiqué dans la référence, change au cours de la préparation du document, ce qui rend très difficile sa mise à jour.

Avant l'avènement de l'informatique, les auteurs évitaient les références internes ; et celles qui étaient inévitables, comme la table des matières (qui, si elle est accompagnée du numéro de page de chaque section, est un exemple de référence interne), étaient écrites à la toute fin.

Même les systèmes de composition les plus limités, comme les traitements de texte, permettent l'inclusion d'une certaine forme de références croisées internes, comme les tables des matières. Mais ce n'est rien comparé au mécanisme complet de gestion des références inclus dans ConTeXt, qui peut également combiner le mécanisme de gestion des références internes visant à maintenir les références à jour, avec l'utilisation d'hyperliens qui n'est évidemment pas exclusive aux références externes.

9.2 Références internes

Deux choses sont nécessaires pour établir une référence interne :

1. Une étiquette ou un identifiant au point cible. Lors de la compilation, ConTeXt, associera des données particulières à cette étiquette. Les données associées dépendent du type d'étiquette ; il peut s'agir du numéro de section, du numéro de note, du numéro d'image, du numéro associé à un élément particulier dans une liste numérotée, du titre de la section, etc.
2. Une commande au point d'origine qui lit les données associées à l'étiquette liée au point cible et les insère au point d'origine. La commande varie en fonction des données de l'étiquette que l'on souhaite insérer au point d'origine.

Lorsque nous pensons à une référence, nous le faisons en termes de « origine → cible », il pourrait donc sembler que les questions relatives à l'origine doivent être expliquées en premier, et ensuite celles relatives à la cible. Cependant, je crois qu'il est plus facile de comprendre la logique des références si l'explication est inversée.

9.2.1 L'étiquette de la cible

Dans ce chapitre, par *étiquette*, j'entends une chaîne de texte qui sera associée au point cible et utilisée en interne pour y faire référence et ainsi récupérer certaines informations concernant le point cible (par exemple le numéro de page, le numéro de section, etc). En fait, les informations associées à chaque étiquette dépendent de la procédure de création de celle-ci. ConTeXt appelle ces étiquettes *références*, mais je pense que ce dernier terme, ayant un sens beaucoup plus large, est moins clair.

L'étiquette associée à la référence cible :

- a besoin que chaque cible potentielle du document soit unique afin qu'elle puisse être identifiée sans aucun doute. Si nous utilisons la même étiquette pour différentes cibles, ConTeXt ne lancera pas d'erreur de compilation (il générera des messages), mais toutes les références pointeront vers la première étiquette qu'il trouvera (dans le fichier source) et cela aura pour effet secondaire que certaines de nos références pourront être erronées et, pire encore, que nous ne les remarquerons pas. Il est donc important de s'assurer, lors de la création d'une étiquette, que la nouvelle étiquette que nous attribuons n'a pas déjà été attribuée auparavant.
- Il peut contenir des lettres, des chiffres, des signes de ponctuation, des espaces vides, etc. Lorsqu'il y a des espaces vides, les règles générales de ConTeXt concernant ces types de caractères s'appliquent toujours (voir [section 4.2.1](#)), de sorte que, par exemple, « **Mon beau libellé** » et « **Mon beau libellé** » sont considérés comme identiques, même si un nombre différent d'espaces vides est utilisé dans les deux.

Puisqu'il n'y a aucune limitation quant aux caractères pouvant faire partie de l'étiquette et à leur nombre, mon conseil est d'utiliser des noms d'étiquette clairs, qui nous aideront à comprendre le fichier source lorsque, peut-être, nous le lirons longtemps après qu'il ait été écrit. C'est pourquoi l'exemple que j'ai donné précédemment (« Mon beau libellé ») n'est pas un bon exemple, car il ne nous dit rien sur la cible vers laquelle pointe l'étiquette. Pour cette rubrique, par exemple, l'étiquette « sec :EtiquettesCibles » serait meilleure (« sec » indiquant qu'il s'agit d'une section).

Pour associer une cible particulière à une étiquette, il existe essentiellement deux procédures :

1. Au moyen d'un argument ou d'une option de commande utilisée pour créer l'élément vers lequel l'étiquette pointera. De ce point de vue, toutes les commandes qui créent une sorte de structure ou d'élément textuel susceptible d'être une cible de référence comprennent une option nommée « `reference` » qui est utilisée pour inclure l'étiquette. Parfois, à la place de l'*option*, l'étiquette est le contenu de l'argument entier.

Nous trouvons un bon exemple de ce que j'essaie de dire dans les commandes de section qui, comme nous le savons depuis (section ??), permettent plusieurs types de syntaxe. Voici comment définir une étiquette avec les deux syntaxes : Dans la syntaxe classique, la commande est écrite comme suit :

```
\section[étiquette]{Titre de la section}

\startsection
  [title=Titre de la section,
   reference=étiquette]
\stopsection
```

Dans les deux cas, la commande prévoit l'introduction d'une étiquette qui sera associée à la section (ou chapter, subsection, etc.) en question.

J'ai dit que cette fonctionnalité se retrouve dans *toutes les commandes* qui nous permettent de créer un élément de texte susceptible d'être la cible d'une référence. Il s'agit de tous les éléments de texte qui peuvent être numérotés, y compris, entre autres, les sections, les objets flottants de toutes sortes (tableaux, images et autres), les notes de bas de page ou de fin, les citations, les listes numérotées, les descriptions, les définitions, etc.

Lorsque l'étiquette est saisie directement avec un argument, et non comme une option à laquelle une valeur est attribuée, il est possible avec ConTeXt d'associer plusieurs étiquettes à une seule cible. Par exemple :

```
\section[étiquette1,étiquette2,étiquette3]{Titre de la section}
```



Je ne vois pas bien quel serait l'avantage d'avoir plusieurs étiquettes différentes pour la même cible et je soupçonne que cela peut être fait non pas parce que cela offre des avantages, mais en raison d'une exigence *interne* de ConTeXt applicable à certains types d'arguments.

2. Au moyen des commandes `\pageref`, `\reference`, ou `\textref` dont la syntaxe est :

Texte initial \pageref{étiquette}
\ref{étiquette}{Texte}
\textref{étiquette}{Texte}
et texte final

Texte initial et texte final

- L'étiquette créée avec `\pageref` nous permet de récupérer le numéro de page.
- Les étiquettes créées avec `\ref` et `\textref` nous permettent de récupérer le numéro de page ainsi que le texte qui leur est associé en argument.

Dans les deux cas `\ref` et `\textref`, le texte lié à l'étiquette n'apparaît pas en tant que tel dans le document final au point où se trouve la commande, mais il peut être récupéré et réapparaître au point d'origine de la référence.

J'ai dit précédemment que chaque étiquette est associée à certaines informations concernant le point cible. La nature de ces informations dépend du type d'étiquette :

- Toutes les étiquettes *rappellent* (dans le sens où elles permettent de les retrouver) le numéro de page de la commande qui les a créées. Pour les étiquettes attachées à des sections qui peuvent avoir plusieurs pages, ce numéro sera celui de la page où commence la section en question.
- Les étiquettes insérées avec la commande qui crée un élément textuel numéroté (section, note, tableau, image, etc.) *rappellent* le numéro associé à cet élément (numéro de section, numéro de note, etc.)
- Si les éléments possèdent un *titre*, comme c'est le cas, par exemple, pour les sections, mais aussi les tableaux s'ils ont été insérés à l'aide de la commande `\placeable`, les étiquettes se *rappelleront* de ces titres.
- Les étiquettes créées avec `\pageref` *rappellent* le numéro de page.
- Celles créées avec `\ref` ou `\textref` *rappellent* également le texte qu'elles prennent comme argument.

En fait, je ne suis pas sûr de la différence réelle entre les commandes `\ref` et `\textref`. Je pense qu'il est possible que la conception des trois commandes qui permettent la création d'étiquettes tente de fonctionner en parallèle avec les trois commandes qui permettent la récupération d'informations à partir des étiquettes (que nous verrons dans un moment) ; mais la vérité est que, selon mes tests, `\ref` et `\textref` semblent être des commandes redondantes.

9.2.2 Commandes au point de référence d'origine pour récupérer les données du point cible

Les commandes que je vais vous expliquer ensuite récupèrent les informations des étiquettes et, en plus, si notre document est interactif, génèrent un lien hypertexte vers la cible. Mais ce qui est important dans ces commandes, c'est l'information qui est récupérée de l'étiquette. Si nous voulons seulement générer l'hyperlien, sans récupérer aucune information de l'étiquette, nous devons utiliser la commande `\goto` expliquée dans la section 9.4.2.

A. Commandes de base pour récupérer les informations d'une étiquette

Sachant que chaque étiquette associée à un point cible peut stocker différents éléments d'information, il est logique que ConTeXt comprenne trois commandes différentes pour récupérer ces informations : selon les informations d'un point cible de référence que nous voulons récupérer, nous utilisons l'une ou l'autre de ces commandes :

- La commande `\at` permet de récupérer le numéro de page de l'étiquette.
- Pour les étiquettes qui mémorisent un numéro d'élément (numéro de section, numéro de note, numéro d'article, numéro de table, etc.) en plus du numéro de page, la commande `\in` permet de récupérer ce numéro.
- Enfin, pour les étiquettes qui mémorisent un texte associé à une étiquette (titre de section, titre d'image inséré avec `\placefigure`, etc.), la commande `\about` permet de récupérer ce texte.

Les trois commandes `\at` `\in` `\about` ont les syntaxes suivantes :

```
\setupinteraction[state=start]
\startsubsection
  [title=Titre de la section,
   reference=sec:cettesection]
\stopsubsection

À la \at{page}[sec:cettesection]
se trouve la \in{section}{.}[sec:cettesection]
dont le titre est \about[sec:cettesection]
```

1 Titre de la section

À la **page 0** se trouve la **section 1**. dont le titre est “**Titre de la section**”

- « `sec:cettesection` » est l'étiquette de la cible dont nous voulons récupérer l'information, c'est un point commun entre toutes les commandes.
- le premier texte entre accolades sera ajouté juste avant l'information que l'on souhaite récupérer avec la commande. Entre le texte et les données de l'étiquette que la commande récupère, un espace insécable sera inséré et si la fonction d'interactivité est activée (avec `\setupinteraction[state=start]`) l'information récupérée par la commande génère un lien hypertexte qui nous dirigera au point cible. Le texte inclus entre accolades fera partie de ce lien (il sera cliquable).

- dans le cas de `\in` il est possible d'indiquer un second texte entre accolades, celui-ci sera ajouté après l'information récupérée par la commande, et sera également intégré au lien hypertexte. Cela est utile par exemple pour générer un lien « 1- Section » au lieu de « Section 1 ».

Ainsi, dans l'exemple précédent, nous voyons comment `\in` récupère le numéro de section et `\at` le numéro de page et `\about` le titre.

Notez que ConTeXt a automatiquement créé des hyperliens (voir section ??), et que le texte pris comme argument par `\in` et `\at` fait partie du lien. Mais si nous l'avions écrit autrement, le résultat serait :

```
\setupinteraction[state=start]
\startsubsection
  [title=Titre de la section,
   reference=sec:cettesection]
\stopsubsection

À la page \at{0}[sec:cettesection]
se trouve la section \in{0-0}[sec:cettesection]
dont le titre est \about[sec:cettesection]
```

1 Titre de la section

À la page 0 se trouve la section 1 dont le titre est “**Titre de la section**”

Le texte reste le même, mais les mots *section* et *page* qui précèdent la référence ne sont pas inclus dans le lien car ils ne font plus partie de la commande.

Si ConTeXt n'est pas en mesure de trouver l'étiquette vers laquelle les commandes `\at`, `\in` ou `\about` pointent, aucune erreur de compilation n'en résultera mais là où les informations récupérées par ces commandes devraient apparaître dans le document final, nous verrons écrit « ?? ».

Il y a deux raisons pour lesquelles ConTeXt ne peut pas trouver une étiquette :

1. Nous avons fait une erreur en l'écrivant.
2. Nous ne compilons qu'une partie du document, et l'étiquette pointe vers la partie non encore compilée (voir sections 4.5.1 et 4.6).

Dans le premier cas, l'erreur devra être corrigée. C'est pourquoi, lorsque nous aurons fini de compiler le document complet (et que le deuxième cas ne sera plus possible), il est bon de rechercher toutes les apparitions de « ?? » dans le PDF pour vérifier qu'il n'y a pas de références cassées dans le document.

B. Récupération des informations associées à une étiquette avec la commande `\ref`

Chacune des commandes `\at`, `\in` et `\about` récupère certains éléments d'une étiquette. Il existe une autre commande qui nous permet de récupérer un élément de l'étiquette indiquée. Il s'agit de la commande `\ref` dont la syntaxe est :

```
\ref [Élément à Récupérer] [Étiquette]
```

où le premier argument peut être :

- `text` : renvoie le texte associé à une étiquette.
- `title` : renvoie le titre associé à un libellé.
- `number` : renvoie le numéro associé à un libellé. Par exemple, dans les sections, le numéro de section.
- `page` : renvoie le numéro de la page.
- `realpage` : renvoie le numéro de la page d'un point de vue physique (car parfois la numérotation des pages ne commence qu'à partir de l'introduction).
- `default` : renvoie ce que ConTEXt considère comme l'élément *natural* de l'étiquette. En général, cela coïncide avec ce qui est renvoyé par `number`.

En fait, `\ref` peut permettre d'être plus précis que `\at`, `\in` ou `\about`, et donc, par exemple, il fait la différence entre le numéro de page et le numéro de page réel. Le numéro de page peut ne pas coïncider avec le numéro réel si, par exemple, la numérotation des pages du document a commencé à 1500 (parce que ce document est la suite d'un précédent) ou si les pages du préambule étaient numérotées en chiffres romains et que, voyant cela, la numérotation a été recommencée. De même, `\ref` fait la différence entre le *text* et le *title* associés à une référence, ce que `\about`, par exemple, ne permet pas.

Si `\ref` est utilisé pour obtenir des informations à partir d'une étiquette qui n'en possède pas (par exemple, le titre d'une étiquette associée à une note de bas de page), la commande renverra une chaîne vide.

```
\setupinteraction[state=start]
\startsubsection
  [title=Titre de la section,
   reference=sec:cettesection]
\stopsubsection

{\tt text :} \ref [text] [sec:cettesection]
\\
{\tt title :} \ref [title] [sec:cettesection]
\\
{\tt number :} \ref [number] [sec:cettesection]
\\
{\tt page :} \ref [page] [sec:cettesection]
\\
{\tt realpage :} \ref [realpage] [sec:cettesection]
```

1 Titre de la section
 text :
 title : Titre de la section
 number : 1
 page : 0
 realpage : 1

C. Déte^cter où le lien mène

ConTEXt possède également deux commandes qui sont sensibles à *l'adresse du lien*. Avec l'« adresse du lien », mon intention est de déterminer si la cible du lien dans le fichier source se trouve avant ou après l'origine. Par exemple : nous sommes en train de rédiger notre document et nous voulons faire référence à une section qui pourrait se trouver avant ou après celle que nous sommes en train d'écrire dans la table des matières finale. Nous n'avons pas encore décidé. Dans cette situation, il serait utile de disposer d'une commande qui écrit l'un ou l'autre selon que la cible se situe finalement avant ou après l'origine dans le document final. Pour des besoins de ce type, ConTEXt fournit la commande `\somewhere` dont la syntaxe est :

```
\somewhere{text si avant}{text si après} [Étiquette].
```

Par exemple :

```
\setupinteraction[state=start]
Dans une \somewhere
  {section précédente} {section prochaine}
  [sec:cettesection].
```



```
\startsubsection
  [title=Titre de la section,
   reference=sec:cettesection]
\stopsubsection

Dans une \somewhere
  {section précédente} {section prochaine}
  [sec:cettesection].
```

Dans une **section prochaine**.

1 Titre de la section

Dans une **section précédente**.

Une autre commande capable de détecter si l'étiquette qu'elle pointe vient avant ou après, est **\atpage** dont la syntaxe est :

```
\atpage[label]
```

Cette commande est assez similaire à la précédente, mais au lieu de nous permettre d'écrire nous-mêmes le texte, selon que l'étiquette se trouve avant ou après, **\atpage** insère un texte par défaut pour chacun des deux cas et, si le document est interactif, insère également un lien hypertexte.

Le texte que **\atpage** insère est celui associé à l'option « `hencefore` » de la commande **\setuplabeltext**, dans le cas où l'étiquette de la cible se trouve *avant* la commande **\atpage** (et celui associé à l'option « `hereafter` » dans le cas contraire).

Lorsque je suis arrivé à ce point, j'ai été trahi par une décision antérieure : dans ce chapitre, j'ai décidé d'appeler ce que ConTeXt appelle une « référence », une « étiquette ». Cela me semblait plus clair. Mais certains fragments de texte générés par les commandes ConTeXt tels que **\atpage**, sont également appelés « labels » (cette fois dans un autre sens). (Voir [section 10.5.3](#)). J'espère que le lecteur ne sera pas perdu par ma façon de présenter. Je pense que le contexte nous permet de distinguer correctement à laquelle des différentes significations de *étiquette* je fais référence dans chaque cas.

Par conséquent, nous pouvons modifier le texte inséré par **\atpage** de la même manière que nous modifions le texte de toute autre étiquette :

```

\mainlanguage[fr]
\setupinteraction[state=start]
\setuplabeltext[fr]
[hereafter=Comme nous le montrerons plus tard]
\setuplabeltext[fr]
[hencefore=Comme nous l'avons vu avant]

\atpage[sec:cettesection].
\startsubsection
  [title=Titre de la section,
   reference=sec:cettesection]
\stopsubsection
\atpage[sec:cettesection].

```

Comme nous le montrerons plus tard.

1 Titre de la section

Comme nous l'avons vu avant.

9.2.3 Génération automatique de préfixes pour éviter les étiquettes en double

Dans un document volumineux, il n'est pas toujours facile d'éviter la duplication des étiquettes. Il est donc conseillé de mettre un peu d'ordre dans la façon dont nous choisissons les étiquettes à utiliser. Une bonne pratique consiste à utiliser des préfixes pour les étiquettes qui varient en fonction du type d'étiquette. Par exemple, « sec : » pour les sections, « fig : » pour les figures, « tbl : » pour les tableaux, etc.

Dans cette optique, ConTeXt inclut une collection d'outils qui permettent :

- de générer automatiquement des étiquettes pour tous les éléments autorisés.
- de faire en sorte que chaque étiquette générée manuellement prenne un préfixe, soit celui que nous avons prédéterminé nous-mêmes, soit celui généré automatiquement par ConTeXt.

L'explication détaillée de ce mécanisme est longue et, bien qu'il s'agisse sans aucun doute d'outils utiles, je ne pense pas qu'ils soient indispensables. Par conséquent, comme ils ne peuvent être expliqués en quelques mots, je préfère ne pas les expliquer et renvoyer à ce qui est dit à leur sujet dans le manuel de référence ConTeXt ou dans le [wiki](#) sur cette question.

9.3 Documents électroniques interactifs

Seuls les documents électroniques peuvent être interactifs, mais tous les documents *électroniques* ne le sont pas. Un document électronique est un document qui est stocké dans un fichier informatique et qui peut être ouvert et lu directement à l'écran. En revanche, un document électronique équipé de fonctionnalités qui permettent à l'utilisateur d'*interagir* avec lui est interactif, c'est-à-dire qu'on peut faire plus que le lire. Il y a interactivité, par exemple, lorsque le document comporte des boutons permettant d'effectuer une action, ou des liens permettant de passer à un autre point du document, ou à un document externe ; ou lorsque le document comporte des zones où l'utilisateur peut écrire, ou des vidéos ou des clips audio qui peuvent être lus, etc.

Tous les documents générés par ConTEXt sont électroniques (puisque ConTEXt génère un PDF qui est par définition un document électronique), mais ils ne sont pas toujours interactifs. Pour leur conférer une interactivité, il est nécessaire de l'indiquer expressément, comme le montre la section suivante.

Il faut cependant savoir que, même si ConTEXt génère un PDF interactif, pour apprécier cette interactivité, nous avons besoin d'un lecteur de PDF capable de le faire, car tous les lecteurs de PDF ne nous permettent pas d'utiliser des hyperliens, des boutons et autres éléments similaires propres aux documents interactifs.

9.3.1 Permettre l'interactivité dans les documents

ConTEXt n'utilise pas de fonctions interactives par défaut, sauf indication expresse, ce qui est normalement fait dans le préambule du document. La commande qui active cet utilitaire est :

```
\setupinteraction[state=start]
```

Normalement, cette commande ne devrait être utilisée qu'une seule fois et dans le préambule du document lorsque nous voulons générer un document interactif. Mais en fait, nous pouvons l'utiliser aussi souvent que nous le souhaitons en modifiant l'état d'interactivité du document. La commande « `state=start` » active l'interactivité, tandis que « `state=stop` » la désactive, de sorte que nous pouvons désactiver l'interactivité dans certains chapitres ou *parties* de notre document comme nous le souhaitons.

Je ne vois aucune raison pour laquelle nous voudrions avoir des parties non interactives dans des documents qui sont interactifs. Mais ce qui est important dans la philosophie de ConTEXt c'est que quelque chose soit techniquement possible, même s'il est peu probable que nous l'utilisions, et elle offre donc une procédure pour le faire. C'est cette philosophie qui donne à ConTEXt tant de possibilités, et empêche une simple introduction comme celle-ci d'être *bref*.

Une fois l'interaction établie :

- Certaines commandes ConTEXt incluent naturellement des hyperliens telles que :

- Les commandes de création de tables des matières : en cliquant sur une entrée de la table des matières, on accède à la page où commence la section en question.
 - Les commandes de références internes que nous avons vues dans la première partie de ce chapitre : cliquer dessus permet de sauter automatiquement à la cible de la référence.
 - Les notes de bas de page et notes de fin de texte où un clic sur l'ancre de la note dans le corps du texte principal nous amènera à la page où la note elle-même est écrite, et un clic sur la marque de la note dans le texte de la note nous amènera au point du texte principal où l'appel a été fait.
 - idem pour les index, etc.
- D'autres commandes sont activées car spécialement conçues pour les documents interactifs, comme les présentations. Celles-ci utilisent de nombreux outils associés à l'interactivité tels que des boutons, des menus, des superpositions d'images, des sons ou des vidéos intégrés, etc. L'explication de tout cela serait trop longue et de plus, les présentations sont un type de document assez particulier. C'est pourquoi, dans les lignes qui suivent, je décrirai une seule fonctionnalité associée à l'interactivité : les hyperliens.

9.3.2 Configuration de base pour les éléments interactifs

`\setupinteraction`, en plus d'activer ou de désactiver l'interaction, nous permet de configurer certains éléments qui y sont liés ; principalement, mais pas seulement, la couleur et le style des liens. Cela se fait par le biais des options de commande suivantes :

- `color` : contrôle la couleur *par défaut* des liens.
- `contrastcolor` : détermine la couleur des liens lorsque leur cible se trouve sur la même page que l'origine. Je recommande que cette option soit toujours définie sur le même contenu que la précédente.
- `style` : contrôle le style du lien.
- `title`, `subtitle`, `author`, `date`, `keyword` : Les valeurs attribuées à ces options seront converties en métadonnées du PDF généré par ConTeXt.
- `click` : Cette option contrôle si le lien doit être mis en évidence lorsqu'il est cliqué.

9.4 Hyperliens vers des documents externes

Je distinguerai les commandes qui ne créent pas le lien mais aident à composer l'URL du lien, et les commandes qui créent l'hyperlien à proprement parler. Examinons-les séparément :

9.4.1 Commandes de définition d'hyperlien (sans les introduire dans le document)

Les URL ont tendance à être très longues, et comprennent des caractères de tous types, même des caractères réservés en ConTeXt et qui ne peuvent pas être utilisés directement. De plus, lorsque l'URL doit être affichée dans le document, il est très difficile de composer le paragraphe, car l'URL peut dépasser la longueur d'une ligne et ne comporte jamais d'espaces vides pouvant être utilisés pour insérer un saut de ligne. De plus, dans une URL, il n'est pas raisonnable de césurer les mots pour insérer des sauts de ligne, car le lecteur pourrait difficilement savoir si la césure fait ou non partie de l'URL.

C'est pourquoi ConTeXt fournit deux utilitaires pour *la composition* des URL. Le premier est principalement destiné aux URL qui seront utilisées en interne, mais qui ne seront pas réellement affichées dans le document. Le second est destiné aux URL qui doivent être écrits dans le texte du document. Examinons-les séparément :

\useURL

Cette commande nous permet d'écrire une URL dans le préambule du document, en l'associant à un nom, de sorte que lorsque nous voulons l'utiliser ensuite dans notre document, nous pouvons l'invoquer par le nom qui lui est associé. Elle est particulièrement utile pour les URL qui seront utilisées plusieurs fois dans le document.

Cette commande permet deux utilisations :

1. \useURL [NomAssocié] [URL]
2. \useURL [NomAssocié] [URL] [] [Texte du lien]

- Dans la première version, l'URL est simplement associée au nom par lequel elle sera invoquée dans notre document. Mais alors, pour utiliser l'URL, nous devrons indiquer d'une manière ou d'une autre, en l'invoquant, quel texte cliquable sera affiché dans le document.

Dans la deuxième version, le dernier argument inclut le texte cliquable. Le troisième argument existe dans le cas où nous voulons diviser une URL en deux parties, de sorte que la première partie contienne l'adresse d'accès et la deuxième partie le nom du document ou de la page spécifique que nous voulons ouvrir. Par exemple, avec l'adresse du document qui explique ce qu'est ConTeXt :

```

\setupinteraction[state=start]
\useURL [WhatIsCTXa]
[http://www.pragma-ade.com/general/manuals/what-is-context.pdf]
[]

[What is \ConTeXt?]

\useURL [WhatIsCTXb]
[http://www.pragma-ade.com/general/manuals/what-is-context.pdf]
[]

[What is \ConTeXt?]

\url[WhatIsCTXa]\\
\url[WhatIsCTXb]

\from[WhatIsCTXa]

```

<http://www.pragma-ade.com/general/manuals/what-is-context.pdf>
<http://www.pragma-ade.com/general/manuals/what-is-context.pdf>
What is ConTeXt?

Dans les deux cas, nous aurons associé cette adresse au mot « `WhatIsCTX` ». Si, à un moment quelconque du texte, nous voulons reproduire une URL que nous avons associée à un nom à l'aide de la commande `\useURL`, nous pouvons utiliser la commande `\url[NomAssocié]` qui insère l'URL associée à ce nom dans le document. Mais cette commande, bien qu'elle écrive l'URL, ne crée aucun lien. La commande `\from` insère le texte associé au lien (voir section 9.4.2).

Le format d'affichage des URL écrites à l'aide de `\url` n'est pas celui établi de manière générale au moyen de `\setupinteraction`, mais celui établi spécifiquement pour cette commande au moyen de `\setupurl`, qui nous permet de définir le style (option `style`) et la couleur (option `color`).

`\hyphenatedurl`

Cette commande est destinée aux URL qui seront écrits dans le texte de notre document, et `ConTeXt` doit inclure des sauts de ligne dans l'URL, si nécessaire, afin de composer correctement le paragraphe. Son format est le suivant :

```
\hyphenatedurl{AdresseURL}
```

Malgré le nom de la commande `\hyphenatedurl`, elle ne met pas de trait d'union dans le nom de l'URL. Ce qu'elle fait, c'est considérer que certains caractères courants dans les URL sont de bons points pour insérer un saut de ligne avant ou après eux. Nous pouvons ajouter les caractères que nous voulons à la liste des caractères pour lesquels un saut de ligne est autorisé. Nous disposons de trois commandes pour cela :

```

\sethyphenatedurlnormal{caractères}
\sethyphenatedurlbefore{caractères}
\sethyphenatedurlafter{caractères}

```

Ces commandes ajoutent, respectivement, les caractères qu'elles prennent comme arguments à la liste des caractères qui supportent les sauts de ligne avant et après, uniquement avant, et uniquement après.

\hyphenatedurl peut être utilisée lorsqu'il faut écrire une URL qui apparaîtra telle quelle dans le document final. Elle peut même être utilisée comme dernier argument de \useURL dans la version de cette commande où le dernier argument récupère le texte cliquable à afficher dans le document final.

Dans l'argument \hyphenatedurl, tous les caractères réservés peuvent être utilisés, sauf trois qui doivent être remplacés par des commandes :

- % doit être remplacé par \letterpercent
- # doit être remplacé par \letterhash
- \ doit être remplacé par \letterescape ou \letterbackslash.

Chaque fois que \hyphenatedurl insère un saut de ligne, il exécute la commande \hyphenatedurlseparator, qui, par défaut, ne fait rien. Mais si nous la redéfinissons, un caractère représentatif est inséré dans l'URL de manière similaire à ce qui se passe avec les mots normaux, où un trait d'union est inséré pour indiquer que le mot continue sur la ligne suivante. Par exemple :

```
\setupinteraction[state=start]
\useURL [WhatIsCTXa]
[http://www.pragma-ade.com/general/manuals]
[what-is-context.pdf]
[http://www.pragma-ade.com/general/manuals/what-is-context.pdf]

\useURL [WhatIsCTXb]
[http://www.pragma-ade.com/general/manuals]
[what-is-context.pdf]
[\hyphenatedurl{http://www.pragma-ade.com/general/manuals/what-is-context.pdf}]

\from[WhatIsCTXa] \\
\from[WhatIsCTXb] \\
\def\hyphenatedurlseparator{\curvearrowright}
\from[WhatIsCTXb]
```

http://www.pragma-ade.com/general/manuals/what-is-context.pdf
http://www.pragma-ade.com/general/manuals/what-is-context.pdf
http://www.pragma-ade.com/general/manuals/what-is-context.pdf

9.4.2 Commandes d'insertion d'hyperlien dans le document

Pour établir des liens vers des URL prédéfinis à l'aide de \useURL, nous pouvons utiliser la commande \from, qui se limite à établir le lien, mais n'écrit aucun texte cliquable. Le texte par défaut de \useURL sera utilisé comme texte du lien. Sa syntaxe est la suivante :

```
\from[Nom]
```

où *Nom* est le nom précédemment associé à une URL à l'aide de \useURL.

```
\setupinteraction[state=start]
\useURL [WhatIsCTX]
  [http://www.pragma-ade.com/general/manuals]
  [what-is-context.pdf]
  [What is \ConTeXt]

\from[WhatIsCTX]
```

What is ConTeXt

Pour créer des liens et les associer à un texte cliquable qui n'a pas été préalablement défini, nous disposons de la commande `\goto` qui est utilisée à la fois pour générer des liens internes et externes. Sa syntaxe est la suivante :

```
\goto{Texte utilisé en lien}[Cible]
```

où *Texte utilisé en lien* est le texte à afficher dans le document final et sur lequel un clic de souris conduira le lecteur à la cible, et *Cible* peut être :

- Une étiquette de notre document. Dans ce cas, `\goto` générera le saut de la même manière que, par exemple, les commandes `\in` ou `\at` déjà examinées. Mais contrairement à ces commandes, aucune information associée à l'étiquette ne sera récupérée.
- L'URL elle-même. Dans ce cas, il faut indiquer expressément qu'il s'agit d'une URL en écrivant la commande comme suit :

```
\goto{Texte utilisé en lien}[url(URL)]
```

où *URL*, à son tour, peut être le nom précédemment associé à une URL au moyen de `\useURL`, ou l'URL elle-même, auquel cas, lors de l'écriture de l'URL, nous devons nous assurer que les caractères réservés de ConTeXt sont écrits correctement dans ConTeXt. L'écriture de l'URL selon les règles de ConTeXt n'affectera pas la fonctionnalité du lien.

9.5 Création de signets dans le PDF final

Les fichiers PDF peuvent avoir une liste de signets internes qui permet au lecteur de voir le contenu du document dans une fenêtre spéciale du programme de visualisation PDF, et de se déplacer dans le document en cliquant simplement sur chacune des sections et sous-sections.

Par défaut, ConTeXt ne fournit pas au PDF de sortie une liste de signets, mais pour qu'il le fasse, il suffit d'inclure la commande `\placebookmarks`, dont la syntaxe est :

```
\placebookmarks [ListeDeSections] [SectionOuverteparDéfaut]
```

où *ListeDeSections* est une liste des niveaux de section, séparées par des virgules, qui doivent apparaître dans la table des matières. *SectionOuverteparDéfaut*, optionnel, indique le niveau de section que le visualisateur de PDF mettre en évidence par défaut (les autres niveaux étant *repliés*). Gardez à l'esprit les observations suivantes concernant cette commande :

- Les types de section définis par l'utilisateur (avec `\definehead`) ne sont pas toujours situés au bon endroit dans la liste des signets. Il est préférable de les exclure.
- Si le titre d'une section comprend une note de fin ou de bas de page, le texte de la note de bas de page est considéré comme faisant partie du signet.
- En guise d'argument, au lieu d'une liste de sections, on peut simplement indiquer le mot symbolique « `all` » qui, comme son nom l'indique, inclura toutes les sections ; cependant, d'après mes tests, ce mot, en plus de ce qui est certainement des sections, inclut des textes placés là avec certaines commandes de non-sectionnement, de sorte que la liste résultante est quelque peu imprévisible.

Tous les programmes de lecture de PDF ne nous permettent pas d'afficher les signets ; et beaucoup de ceux qui le font n'ont pas cette fonction activée par défaut. Par conséquent, pour vérifier le résultat de cette fonction, nous devons nous assurer que notre programme de lecture de PDF prend en charge cette fonction et qu'elle est activée. Je crois me souvenir qu'Acrobat, par exemple, n'affiche pas les signets par défaut, bien qu'il existe un bouton dans sa barre d'outils pour les afficher.

9.6 Pièces jointes

Une autre fonctionnalité d'interactivité de ConTEXt : il permet d'introduire des pièces attachées dans vos documents avec la commande `\attachment`.

```
\attachment  
  [Nom de la pièce jointe]  
  [file={fichier.txt},  
   author={auteur de la pièce jointe}]
```

Le fichier attaché doit être dans le même répertoire que le fichier `.tex` dans lequel se trouve la commande `\attachment` ou bien être dans l'un des répertoires indiqués avec la commande `\usepath` (voir section 4.6.4).

9.7 Références bibliographiques

Partons ici de l'hypothèse que vous avez un fichier BiBTeX .bib qui contient les références bibliographiques. Les principales commandes à connaître sont : `\usebtxdataset`, `\showbtxdatasetfields`, `\cite` et `\placelistofpublications`. Voyez plutôt :



```
\usebtxdataset [introCTX_biblio.bib]%
\showbtxdatasetfields%
%\showbtxdatasetcompleteness
```

Une première citation~: `\cite[title]{Hagen2017metafun}`
de `\cite[author]{Hagen2017metafun} (\cite[Hagen2017metafun])`

Une seconde citation~: `\cite[title]{ConTeXtExcursion}`
de `\cite[author]{ConTeXtExcursion} (\cite[ConTeXtExcursion])`

```
\startsubject[title=Biblio]
\placelistofpublications
\stopsubject
```

tag	category	fields
ConTeXtExcursion	book	author title url year
ConTeXtIntroductionFr	book	author language title url year
Hagen2017metafun	book	abstract author day edition file isbn keywords language month organization publisher title totalpages url year

Une première citation : *MetaFun* de Hagen ([1])

Une seconde citation : *ConTeXt an excursion* de Otten ([2])

Biblio

1 H. Hagen, *MetaFun* (4 ed.), Boekplan, NL, 2017.

2 T. Otten, *ConTeXt an excursion*, 2017.

Une première citation : *MetaFun* de Hagen ([1])

Une seconde citation : *ConTeXt an excursion* de Otten ([2])

Cette section méritera d'être complétée. Pour le moment, vous pouvez consulter la présentation de 2014 de Alan Braslau, ainsi que le manuel officiel *Bibliographies the ConTeXt way* qui indique notamment comment personnaliser l'affichage des informations.

Biblio

- 1 H. Hagen, *MetaFun* (4 édition), Boekplan, NL, 2017.
- 2 T. Otten, *ConTeXt an excursion*, 2017.

III

Composition des éléments locaux

Chapitre 10

Caractères, mots, texte et espace horizontal

Table of Contents : [10.1 Obtenir des caractères qui ne sont pas normalement accessibles à partir du clavier](#) ; [10.1.1 Diacritiques et lettres spéciales](#) ; [10.1.2 Ligatures traditionnelles](#) ; [10.1.3 Alphabet grec](#) ; [10.1.4 Divers symboles](#) ; [10.1.5 Définir un caractère](#) ; [10.1.6 Utilisation de jeux de symboles prédéfinis](#) ; [10.2 Formatages spéciaux des caractères](#) ; [10.2.1 Majuscules, minuscules et fausses petites majuscules](#) ; [10.2.2 Lettrine](#) ; [10.2.3 Texte en exposant ou en indice](#) ; [10.2.4 Texte en verbatim](#) ; [10.3 Espacement des caractères et des mots](#) ; [10.3.1 Réglage automatique de l'espacement horizontal](#) ; [10.3.2 Modification de l'espace entre les caractères et les mots](#) ; [10.3.3 Commandes pour ajouter un espace horizontal entre les mots](#) ; [10.4 Mots composés](#) ; [10.5 La langue du texte](#) ; [10.5.1 Définition et modification de la langue](#) ; [10.5.2 Configuration de la langue](#) ; [10.5.3 Étiquettes associées à des langues particulières](#) ; [10.5.4 Quelques commandes liées à la langue](#) ; [A Commandes liées à la date](#) ; [B La commande \translate](#) ; [C Les commandes \quote et \quotation](#) ;

L'élément de base de tous les documents textuels est le caractère : les caractères sont regroupés en mots, qui à leur tour forment des lignes qui constituent les paragraphes qui composent les pages.

Le présent chapitre, qui commence par « *caractère* », explique certains des utilitaires de ConTeXt relatifs aux caractères, aux mots et au texte.

10.1 Obtenir des caractères qui ne sont pas normalement accessibles à partir du clavier

Dans un fichier texte codé en UTF-8 (voir section ??), nous pouvons utiliser n'importe quel caractère ou symbole, aussi bien des langues vivantes que de nombreuses langues dites mortes. Mais, comme les possibilités d'un clavier sont limitées, la plupart des caractères et symboles autorisés en UTF-8 ne peuvent normalement pas être obtenus directement du clavier. C'est notamment le cas de nombreux diacritiques, c'est-à-dire des signes placés au-dessus (ou au-dessous) de certaines lettres, leur conférant une valeur particulière ; mais aussi de nombreux autres caractères comme les symboles mathématiques, les ligatures traditionnelles, etc. Nous pouvons obtenir beaucoup de ces caractères avec ConTeXt en utilisant des commandes.

10.1.1 Diacritiques et lettres spéciales

¹⁶ Parmi les commandes trouvées dans [table 10.1](#) le tilde ne fonctionne pas avec la lettre «*e*», et je ne sais pas pourquoi.

¹⁷ Nous rappelons que dans ce document, nous représentons les espaces vides, lorsqu'il est important de les voir, par le «*u*».

Presque toutes les langues occidentales ont des diacritiques (à l'exception importante de l'anglais pour l'essentiel) et en général, les claviers peuvent générer les diacritiques correspondant aux langues régionales. Ainsi, un clavier espagnol peut générer tous les signes diacritiques nécessaires à l'espagnol (essentiellement les accents et le tréma) ainsi que certains signes diacritiques utilisés dans d'autres langues comme le catalan (accents graves et cédilles) ou le français (cédilles, accents graves et circonflexes) ; mais pas, par exemple, certains signes diacritiques utilisés en portugais, comme le tilde sur certaines voyelles dans des mots comme « *navegação* ».

TeX a été conçu aux États-Unis où les claviers ne permettent généralement pas d'obtenir les diacritiques ; Donald Knuth lui a donc donné un ensemble de commandes qui permettent d'obtenir presque tous les diacritiques connus (du moins dans les langues utilisant l'alphabet latin). Si nous utilisons un clavier espagnol, cela n'a pas beaucoup de sens d'utiliser ces commandes pour obtenir les diacritiques qui peuvent être obtenus directement à partir du clavier. Il est néanmoins important de savoir que ces commandes existent, et ce qu'elles sont, puisque les claviers espagnols (ou italiens, ou français...) ne permettent pas de générer tous les diacritiques possibles.

Nom	Caractère	Abréviation	Commande
Accent aiguë	ú	\`{u}	\uacute
Accent grave	ù	\`{u}	\ugrave
Accent circonflexe	û	\^{u}	\ucircumflex
Accent tréma	ü	\^{u}	\udiaeresis ou \uumlaut
Tilde	\~{u}	\~{u}	\utilde
Macron	\={u}	\={u}	\umacron
Brève	\u{u}	\u{u}	\ubreve

Tableau 10.1 Accents and other diacritics

Dans la [table 10.1](#) nous trouvons les commandes et abréviations qui nous permettent d'obtenir ces diacritiques. Dans tous les cas, il importe peu que nous utilisions la commande ou l'abréviation. Dans le tableau, j'ai utilisé la lettre « *u* » comme exemple, mais ces commandes fonctionnent avec n'importe quelle voyelle (la plupart d'entre elles¹⁶) et aussi avec certaines consonnes et certaines semi-voyelles.

- Comme la plupart des commandes abrégées sont des *symboles de contrôle* (voir [section 3.2](#)), la lettre sur laquelle le diacritique doit tomber peut être écrite immédiatement après la commande, ou séparée de celle-ci. Ainsi, par exemple, pour obtenir le « *ã* » portugais, on peut écrire les caractères \lettertilde a ou \lettertilde\textvisiblespace a.¹⁷ Mais dans le cas du brève (\u), lorsqu'il s'agit d'un *mot de contrôle* l'espace blanc est obligatoire.
- Dans le cas de la version longue de la commande, la lettre sur laquelle tombe le diacritique sera la première lettre du nom de la commande. Ainsi, par exemple, \emacron placera un macron au-dessus d'un « *e* » minuscule (é), \Emacron fera de même au-dessus d'un « *E* » majuscule (É), tandis que \Amacron fera de même au-dessus d'un « *A* » majuscule (Á).

Alors que les commandes de [table 10.1](#) fonctionnent avec les voyelles et certaines consonnes, il existe d'autres commandes pour générer certains diacritiques et lettres spéciales qui ne fonctionnent que sur une ou plusieurs lettres. Elles sont présentées dans [table 10.2](#).

Nom	Caractère	Abréviation	Commande
O scandinave	ø, Ø	\o, \O	
O scandinave	å, Å, Å	\aa, \AA, \r a, \r A	\aring, \Aring
L polonais	ł, Ł	\l, \L	
Eszett allemand	ß, ß	\sz, \ss	
« i » et « j » sans point	ı, į	\i, \j	
Umlaut (ou tréma) hongrois,	ő, Ő	\H u, \H U	
Cedilla	ç, Ç	\c c, \c C	\ccedilla, \Ccedilla

Tableau 10.2 More diacritics and special letters

Je tiens à préciser que certaines des commandes du tableau ci-dessus génèrent les caractères à partir d'autres caractères, tandis que d'autres commandes ne fonctionnent que si la police que nous utilisons a expressément prévu le caractère en question. Ainsi, en ce qui concerne l'Eszett allemand (ß), le tableau indique deux commandes mais un seul caractère, car la police que j'utilise ici pour ce texte ne prévoit que la version majuscule de l'eszett allemand (ce qui est assez courant).

C'est probablement la raison pour laquelle je n'arrive pas non plus à obtenir le A scandinave en majuscules, bien que « {\r A} » et `\Aring` fonctionnent correctement.

Le tréma hongrois fonctionne également avec la lettre « o », et la cédille avec les lettres « k », « l », « n », « r », « s » et « t », respectivement en minuscule ou en majuscule. Les commandes à utiliser sont respectivement `\kcedilla`, `\lcedilla`, `\ncedilla`...

10.1.2 Ligatures traditionnelles

Une ligature est formée par l'union de deux ou plusieurs graphèmes qui s'écrivent habituellement séparément. Cette « fusion » entre deux caractères a souvent commencé comme une sorte de sténographie dans les textes manuscrits, pour finalement atteindre une certaine indépendance typographique. Certains d'entre eux ont même été inclus parmi les caractères habituellement définis dans une police typographique, comme l'esperluette, « & », qui a commencé comme une contraction de la copule (conjonction) latine « et », ou l'Eszett (ß) allemand, qui, comme son nom l'indique, a commencé comme une combinaison d'un « s » et d'un « z ». Dans certaines polices de caractères, même aujourd'hui, nous pouvons retrouver les origines de ces deux caractères ; ou peut-être que je les vois parce que je sais qu'ils sont là. En particulier, avec la police Pagella pour « & » et avec Bookman pour « ß ».

Comme exercice, je suggère (après avoir lu le chapitre ?, où il explique comment faire) d'essayer de représenter ces caractères avec ces polices à une taille suffisamment grande (par exemple, 30 pt) pour pouvoir travailler sur leurs composants.

¹⁸ Dans L^AT_EX, par contre, nous pouvons utiliser la commande `\DH` implémentée par le paquet « `fontenc` ».

D'autres ligatures traditionnelles qui ne sont pas devenues aussi populaires, mais qui sont encore utilisées occasionnellement aujourd'hui, sont les terminaisons latines « oe » et « ae » qui étaient parfois écrites comme « œ » ou « ae » pour indiquer qu'elles formaient une diphthongue en latin. Ces ligatures peuvent être réalisées dans ConTeXt avec les commandes trouvées dans [table 10.3](#)

Ligature	Abréviation	Commande
æ, Æ	\ae, \AE	\aeligature, \AEligature
œ, œ	\oe, \OE	\oeligature, \OEligature

Tableau 10.3 Ligatures traditionnelles

Une ligature qui était autrefois traditionnelle en espagnol (castillan) et que l'on ne trouve généralement pas dans les polices actuelles est « Ð » : une contraction impliquant « D » et « E ». Pour autant que je sache, il n'y a pas de commande dans ConTeXt qui nous permette de l'utiliser,¹⁸ mais nous pouvons en créer un, comme expliqué dans la section [sec:create characters].

En plus des ligatures précédentes, que j'ai appelées *traditionnelles* parce qu'elles proviennent de l'écriture manuscrite, après l'invention de la presse à imprimer, certaines ligatures de texte imprimé se sont développées, que j'appellerai « ligatures typographiques », considérées par ConTeXt comme des fonctionnalités de police et qui sont gérées automatiquement par le programme, bien que nous puissions influencer la façon dont ces fonctionnalités de police sont gérés (y compris les ligatures) avec `\definefontfeature` (voir [section 6.7.5](#))

10.1.3 Alphabet grec

Il est courant d'utiliser des caractères grecs dans les formules mathématiques et physiques. C'est pourquoi ConTeXt a inclus la possibilité de générer tout l'alphabet grec, majuscules et minuscules. Ici, la commande est construite sur le nom anglais de la lettre grecque en question. Si le premier caractère est écrit en minuscule, nous aurons la lettre grecque en minuscule et s'il est écrit en majuscule, nous aurons la lettre grecque en majuscule. Par exemple, la commande `\mu` générera la version minuscule de cette lettre (μ) tandis que `\Mu` générera la version majuscule (M). Dans la [table 10.4](#), nous pouvons voir quelle commande génère chacune des lettres de l'alphabet grec, en minuscule et en majuscule.

Notez que pour les versions minuscules de certains caractères (epsilon, kappa, theta, pi, rho, sigma et phi), il existe deux variantes possibles.

10.1.4 Divers symboles

Outre les caractères que nous venons de voir, T_EX (et donc ConTeXt) propose des commandes permettant de générer un nombre quelconque de symboles. Ces commandes sont nombreuses. J'en ai fourni une liste étendue mais incomplète dans [appendix B](#).

Nom	Caractère (min/maj)	Commande (lc/uc)
Alpha	α, A	<code>\alpha, \Alpha</code>
Bêta	β, B	<code>\beta, \Beta</code>
Gamma	γ, Γ	<code>\gamma, \Gamma</code>
Delta	δ, Δ	<code>\delta, \Delta</code>
Epsilon	ϵ, ε, E	<code>\epsilon, \varepsilon, \Epsilon</code>
Zéta	ζ, Z	<code>\zeta, \Zeta</code>
Êta	η, H	<code>\eta, \Eta</code>
Thêta	$\theta, \vartheta, \Theta$	<code>\theta, \vartheta, \Theta</code>
Iota	ι, I	<code>\iota, \Iota</code>
Kappa	κ, \varkappa, K	<code>\kappa, \varkappa, \Kappa</code>
Lambda	λ, Λ	<code>\lambda, \Lambda</code>
Mu	μ, M	<code>\mu, \Mu</code>
Nu	ν, N	<code>\nu, \Nu</code>
Ksi ou Xi	ξ, Ξ	<code>\xi, \Xi</code>
Omicron	\omicron, O	<code>\omicron, \Omicron</code>
Pi	π, ω, Π	<code>\pi, \varpi, \Pi</code>
Rhô	ρ, ϱ, P	<code>\rho, \varrho, \Rho</code>
Sigma	$\sigma, \varsigma, \Sigma$	<code>\sigma, \varsigma, \Sigma</code>
Tau	τ, T	<code>\tau, \Tau</code>
Upsilon	υ, Y	<code>\upsilon, \Upsilon</code>
Phi	ϕ, φ, Φ	<code>\phi, \varphi, \Phi</code>
Khi ou Chi	χ, X	<code>\chi, \Chi</code>
Psi	ψ, Ψ	<code>\psi, \Psi</code>
Oméga	ω, Ω	<code>\omega, \Omega</code>

Tableau 10.4 Alphabet grec

10.1.5 Définir un caractère

Si nous devons utiliser des caractères qui ne sont pas accessibles depuis notre clavier, nous pouvons toujours trouver une page Web avec ces caractères et les copier dans notre fichier source. Si nous utilisons l'encodage UTF-8 (comme recommandé), cela fonctionnera presque toujours (**si la police que vous utilisez les contient**, comme DejaVu qui les contient presque tous). Mais il existe également sur le wiki ConTEXt [une page contenant des tas de symboles](#) qui peuvent être simplement copiés et collés dans notre document.

Toutefois, si nous devons utiliser plusieurs fois l'un des caractères en question, le copier-coller n'est pas le moyen le plus efficace d'y parvenir. Il serait préférable de définir le caractère de façon à ce qu'il soit associé à une commande qui le générera à chaque fois. Pour ce faire, on utilise `\definecharacter` dont la syntaxe est :

```
\definecharacter {Nom} {Caractère}
```

où

- **Nom** est le nom associé au nouveau caractère. Il ne doit pas s'agir du nom d'une commande existante, car cela écraserait cette commande.
- **Caractère** est le caractère généré chaque fois que nous exécutons `\Nom`. Il existe trois façons d'indiquer ce caractère :

- en l'écrivant simplement ou en le collant dans notre fichier source (si nous l'avons copié d'un autre document électronique ou d'une page Web).
- en indiquant le numéro associé à ce caractère dans la police que nous utilisons actuellement. Pour voir les caractères inclus dans la police et les numéros qui leur sont associés, nous pouvons utiliser la commande `\showfont[\em NomDePolice]` (voir section).
- Construire le nouveau caractère avec l'une des commandes de construction de caractères composites que nous verrons immédiatement après.

Pour illustrer la première utilisation, revenons pour l'instant aux sections traitant des ligatures (10.1.2). J'y ai parlé d'une ligature traditionnelle en espagnol que l'on ne trouve généralement pas dans les polices actuelles : « Ð ». Nous pourrions associer ce caractère, par exemple, à la commande `\decontract` afin que le caractère soit généré chaque fois que nous écrivons `\decontract`. Nous faisons cela avec :

```
\definecharacter {decontract} {Ð}
Ce caractère~: \tfd \decontract.
```

Ce caractère : Ð.

Pour construire un nouveau caractère qui n'est pas dans notre police, et qui ne peut pas être obtenu à partir du clavier, comme c'est le cas de l'exemple que je viens de donner, il faut d'abord trouver un texte où ce caractère se trouve, le copier et être capable de le coller dans notre définition. Dans l'exemple que je viens de donner, j'ai initialement copié la « Ð » de Wikipedia.

ConTeXt comprend également certaines commandes qui nous permettent de créer des caractères composites et qui peuvent être utilisées en combinaison avec `\definecharacter`. Par caractères composites, j'entends les caractères qui possèdent également des diacritiques. Les commandes sont les suivantes :

```
\definecharacter zete      {\buildtextaccent `z}
\definecharacter zcomma   {\buildtextbottomcomma z}
\definecharacter zdot     {\buildtextbottomdot z}
\definecharacter zcedilla {\buildtextcedilla z}
\definecharacter zgrave   {\buildtextgrave z}
\definecharacter zmacron  {\buildtextmacron z}

zete      ~,~ zcomma   ~,~ zdot     ~,~
zcedilla ~,~ zgrave   ~,~ zmacron
```

zete , zcomma , zdot , zcedilla , zgrave , zmacron

Par exemple : comme nous le savons déjà, par défaut, ConTeXt ne possède que des commandes pour écrire certaines lettres avec une cédille (c, k, l, n, r, s y t) qui sont généralement incorporées dans les polices de caractères. Si nous voulions utiliser un « b », nous pourrions utiliser la commande `\buildtextcedilla` comme dans l'exemple.

Cette commande crée la nouvelle commande `\bB` qui générera un « b » avec une cédille. Ces commandes construisent littéralement le nouveau caractère qui sera généré même si notre police ne le possède pas. Ce que font ces commandes, c'est superposer un caractère sur un autre, puis donner un nom à cette superposition.

Lors de mes tests, je n'ai pas réussi à faire fonctionner `\buildmathaccent` ou `\buildtex-tognek`. Je ne les mentionnerai donc plus à partir de maintenant.

`\buildtextaccent` prend deux caractères comme arguments et en superpose un sur l'autre, en relevant légèrement l'un d'eux. Bien qu'elle soit appelée « buildtex-taccent », il n'est pas essentiel que l'un des caractères pris comme arguments soit un accent ; mais la superposition donnera de meilleurs résultats s'il l'est, car dans ce cas, en superposant l'accent sur le caractère, l'accent a moins de chances d'écraser le caractère. D'autre part, le chevauchement de deux caractères qui ont la même ligne de base dans des conditions normales est affecté par le fait que la commande élève légèrement l'un des caractères au-dessus de l'autre. C'est pourquoi nous ne pouvons pas utiliser cette commande, par exemple, pour obtenir la contraction « Ð » mentionnée plus haut, car si nous écrivons :

```
\definecharacter {decontract}
{\buildtextaccent D E}
\decontract ~ vs ~ Ð
```

Ð vs Ð

dans notre fichier source, la légère élévation au-dessus de la ligne de base de « Ð » que cette commande produit n'est pas très bon. Mais si la hauteur des caractères le permet, nous pourrions créer une combinaison. Par exemple :

```
\definecharacter unusual {\buildtextaccent
  regarde : "
\_
"}  
regarde~: \unusual
```

Le reste des commandes de construction prend un seul argument – le caractère auquel le diacritique généré par chaque commande sera ajouté. L'exemple ci-dessus montre un exemple de chacune d'entre elles, construite sur la lettre « z » :

- `\buildtextbottomcomma` ajoute une virgule sous le caractère qu'il prend comme argument.
- `\buildtextbottomdot` ajoute un point sous le caractère qu'il prend comme argument.
- `\buildtextcedilla` ajoute une cédille sous le caractère qu'il prend comme argument.
- `\buildtextgrave` ajoute un accent grave sur le caractère qu'il prend comme argument.
- `\buildtextmacron` ajoute une petite barre sous le caractère qu'il prend comme argument.

À première vue, `\buildtextgrave` semble redondant étant donné que nous avons `\buildtextaccent` ; Cependant, si vous vérifiez l'accent grave généré par la première de ces deux commandes, il semble un peu mieux. L'exemple suivant montre le résultat des deux commandes, à une taille de police suffisante pour apprécier la différence :

```
\definecharacter zgrave {\buildtextgrave z}
\definecharacter zgraveb {\buildtextaccent `z}
\switchtobodyfont[30pt]
\midaligned{\framed{\zgrave\ -- \zgraveb}}
```

10.1.6 Utilisation de jeux de symboles prédéfinis

« ConTeXt Standalone » comprend, en plus de ConTeXt un certain nombre de jeux de symboles prédéfinis que nous pouvons utiliser dans nos documents. Ces ensembles sont appelés « cc », « cow », « fontawesome », « jmn », « mvs » et « nav ». Chacun de ces ensembles comprend également des sous-ensembles :

- `cc` comprends « cc ».
- `cow` comprends « cownormal » et « cowcontour ».
- `fontawesome` comprends « fontawesome-regular », « fontawesome-solid » et « fontawesome-brands ».
- `jmn` comprends « navigation 1 », « navigation 2 », « navigation 3 » et « navigation 4 ».
- `mvs` comprends « astronomic », « zodiac », « europe », « martin vogel 1 », « martin vogel 2 » et « martin vogel 3 ».
- `nav` comprends « navigation 1 », « navigation 2 » et « navigation 3 ».

Le wiki mentionne également un ensemble appelé `was` qui inclut « wasy general », « wasy music », « wasy astronomy », « wasy astrology », « wasy geometry », « wasy physics » et « wasy apl ». Mais je n'ai pas pu les trouver dans ma distribution, et mes tests pour tenter d'y accéder ont échoué.

Pour voir les symboles spécifiques contenus dans chacun de ces ensembles, on utilise les commandes `\usesymbols \showsymbolset`. Par exemple : si nous voulons voir les symboles inclus dans « `mvs/zodiac` », alors dans le fichier source nous devons écrire :

```
\usesymbols [mvs]
\showsymbolset[zodiac]
```

Aquarius
Aries
Cancer
Capricorn
Gemini
Leo
Libra
Pisces
Sagittarius
Scorpio
Taurus
Virgo

Notez que le nom de chaque symbole est indiqué ainsi que le symbole. La commande `\symbol` nous permet alors d'utiliser n'importe lequel des symboles. Par exemple, si nous voulons utiliser le symbole astrologique associé au Verseau (trouvé dans mvs/zodiac), nous devons écrire :

```
\usesymbols [mvs]
\symbol[zodiac] [Aquarius]
```

Le symbole est traité comme un « caractère » et sera donc affectée par la taille de police active lors de l'impression. Nous pouvons également utiliser `\definecharacter` pour associer le symbole en question à une commande. Par exemple et utiliser ces symboles, par exemple, dans un environnement d'énumération. Par exemple :

```
\usesymbols [mvs]
\definecharacter %
  SymbAries \symbol[zodiac] [Aquarius]
\definesymbol [1]
  [\symbol[martin vogel 2] [PointingHand]]
\definesymbol [2]
  [\symbol[martin vogel 2] [CheckedBox]]

\startitemize[packed]
  \item item avec \SymbAries
  \item item
\startitemize[packed]
  \item item
  \item item
\stopitemize
  \item item
\stopitemize
```

item avec
item
item
item
item

10.2 Formatages spéciaux des caractères

À proprement parler, ce sont les commandes de *formatage* qui affectent la police utilisée, sa taille, son style ou sa variante. Ces commandes sont expliquées dans le Chapitre ???. Cependant, d'un point de vue plus *général*, nous pouvons également considérer les commandes qui modifient d'une manière ou d'une autre les caractères qu'elles prennent comme argument (modifiant ainsi leur apparence) comme des commandes de formatage du texte. Nous allons examiner certaines de ces commandes dans cette section. D'autres, telles que le texte souligné ou aligné avec des lignes au-dessus ou au-dessous du texte courant (par exemple, lorsque nous voulons fournir de l'espace pour répondre à une question) seront vues dans [section 12.5](#).

10.2.1 Majuscules, minuscules et fausses petites majuscules

Les lettres elles-mêmes peuvent être en majuscules ou en minuscules. Pour ConTeXt, les majuscules et les minuscules sont des caractères différents, donc en principe il composera les lettres comme il les trouve écrites. Cependant, il existe un groupe de commandes qui nous permettent de nous assurer que le texte qu'elles prennent comme argument est toujours écrit en majuscules ou en minuscules :

- `\word{texte}` : convertit le texte pris comme argument en minuscules.
- `\Word{texte}` : convertit en majuscule la première lettre du texte pris en argument.
- `\Words{texte}` : convertit en majuscule la première lettre de chacun des mots pris comme argument ; les autres sont en minuscule.
- `\WORD{texte}` ou `\WORD{texte}` : écrit le texte pris comme argument en majuscules.

```
\word      {ceCi esT UN tEST} \\  
\Word     {ceCi esT UN tEST} \\  
\Words    {ceCi esT UN tEST} \\  
\WORD     {ceCi esT UN tEST} \\  
\WORDS    {ceCi esT UN tEST} \\  
\Word{\word {ceCi esT UN tEST}}
```

```
ceci est un test  
ceci est un test
```

Les commandes `\cap` et `\Cap` sont très similaires à celles-ci : elles mettent également en majuscules le texte qu'elles prennent comme argument, mais lui appliquent ensuite un facteur d'échelle égal à celui appliqué par le suffixe « *x* » dans les commandes de changement de police (voir [section 6.4.2](#)) de sorte que, dans la plupart des polices, les majuscules auront la même hauteur que les minuscules, ce qui nous donne une sorte d'effet de fausses petites capitales. Par rapport aux vraies petites capitales (voir [section 6.5.2](#)), celles-ci présentent les avantages suivants :

item `\cap` et `\Cap` fonctionnent avec n'importe quelle police, contrairement aux vraies petites capitales qui ne fonctionnent qu'avec les polices et les styles qui les incluent expressément.

- Les vraies petites capitales, quant à elles, sont une variante de la police qui, en tant que telle, est incompatible avec toute autre variante telle que le gras, l'italique ou l'incliné. En revanche, `\cap` et `\Cap` sont entièrement compatibles avec toutes les variantes de police.

La différence entre `\cap` et `\Cap` est que, tandis que le premier applique le facteur d'échelle à toutes les lettres des mots qui composent son argument, `\Cap` n'applique aucune échelle à la première lettre de chaque mot, obtenant ainsi un effet similaire à celui que l'on obtient si l'on utilise de vraies majuscules dans un texte en petites capitales. Si le texte pris comme argument dans « caps » est constitué de plusieurs mots, la taille de la majuscule de la première lettre de chaque mot sera maintenue.

L'ONU, dont le `\Cap{président}` a son bureau au siège de l'`\cap{oNu}`...

L'ONU, dont le `\Cap{président}` a son bureau au siège de l'ONU...

Il faut noter, tout d'abord, la différence de taille entre la première fois que nous écrivons « ONU » (en majuscules) et la deuxième fois (en petites capitales, « ONU »). Dans l'exemple, j'ai écrit `\cap{oNu}` la deuxième fois pour que nous puissions voir que cela n'a pas d'importance si nous écrivons l'argument que prend `\cap` en majuscules ou en minuscules : la commande convertit toutes les lettres en majuscules et applique ensuite un facteur de mise à l'échelle, contrairement à `\Cap` qui ne met pas à l'échelle la première lettre.

Ces commandes peuvent également être *imbriquées*, auquel cas le facteur d'échelle est appliqué une fois de plus, ce qui entraîne une réduction supplémentaire, comme dans l'exemple suivant où le mot « capital » de la première ligne est encore réduit :

Les Personnes, `\cap{Les personnes qui ont amassé leur \cap{capital} au détriment des autres sont le plus souvent {\bf décapitées} en période de révolution}`.

Les Personnes, LES PERSONNES QUI ONT AMASSÉ LEUR CAPITAL AU DÉTRIMENT DES AUTRES SONT LE PLUS SOUVENT DÉCAPITÉES EN PÉRIODE DE RÉVOLUTION.

La commande `\nocap` appliquée à un texte auquel est appliqué `\cap`, annule l'effet de `\cap` dans le texte qui est son argument. Par exemple :

`\cap{Quand j'étais un, je venais de commencer, quand j'étais deux, j'étais \nocap{presque} nouveau (A.A. Milne)}`.

QUAND J'ÉTAIS UN, JE VENAISS DE COMMENCER, QUAN J'ÉTAIS DEUX, J'ÉTAIS presque NOUVEAU (A.A. MILNE).

Nous pouvons configurer le fonctionnement de `\cap` avec `\setupcapitals` et nous pouvons également définir différentes versions de la commande, chacune avec son propre nom et sa configuration spécifique. Nous pouvons le faire avec `\definecapitals`.

Les deux commandes fonctionnent de manière similaire :

```
\definecapitals [Nom] [Configuration]
\setupcapitals [Nom] [Configuration]
```

Le paramètre « Nom » de la commande `\setupcapitals` est facultatif. S'il n'est pas utilisé, la configuration affectera la commande `\cap` elle-même. S'il est utilisé, nous devons donner le nom que nous avons précédemment attribué dans `\definecapitals` à une configuration réelle.

Dans l'une ou l'autre des deux commandes, la configuration permet trois options : « title », « sc » et « style », la première et la seconde autorisant « yes » et « no » comme valeurs. Avec « title », nous indiquons si la capitalisation affectera également les titres (ce qui est le cas par défaut) et avec « sc », nous indiquons si la commande doit utiliser de véritables petites capitales (« yes ») ou de fausses petites capitales (« no »). Par défaut, il utilise les fausses petites capitales, ce qui présente l'avantage que la commande fonctionne même si vous utilisez une police qui n'a pas implémenté les petites capitales. La troisième valeur « style » permet d'indiquer une commande de style à appliquer au texte affecté par la commande `\cap`.

10.2.2 Lettrine

Une lettrine est une lettre initiale majuscule décorée placée en tête d'un texte et occupant une hauteur supérieure à la ligne courante. Le sujet n'est abordé ici que pour faire prendre connaissance des commandes `\setupinitial` et `\placeinitial`.

```
\setupinitial [...] 1 [...] 2 [...]
OPT
1 NAME
2 n      = NUMBER
m      = NUMBER
before = COMMAND
distance = DIMENSION
hoffset = DIMENSION
voffset = line DIMENSION
style   = STYLE COMMAND
color   = COLOR
font    = FONT
text    = TEXT
location = margin text
method   = first last auto none
```

```
\setupinitial
[color=darkred,
distance=0.75em,
n=3,
hoffset=-0.25em]
```

```
\placeinitial Bon, c'est simple.
Vous suivez les baffes. Et il
devrait être au bout. J'veux
accompagne pas hein, j'veais rester
là et réfléchir un peu
```

Bon, c'est simple. Vous suivez les baffes. Et il devrait être au bout. J'veux accompagne pas hein, j'veais rester là et réfléchir un peu

10.2.3 Texte en exposant ou en indice

Nous savons déjà (voir section 3.1) que dans le mode maths, les caractères réservés « `_` » et « `^` » convertiront le caractère ou le groupe qui suit immédiatement en exposant ou en indice. Pour obtenir cet effet en dehors du mode mathématique, ConTEXt comprend les commandes suivantes :

- `\high{Texte}` : écrit le texte qu'il prend comme argument en exposant.
- `\low{Texte}` : écrit le texte qu'elle prend comme argument en indice.
- `\lohi{texte 1}{Texte 2}` : écrit les deux arguments l'un au-dessus de l'autre : en bas le premier argument, et en haut le second.

Ceci est un texte `\high{en haut}`,
un autre `\low{en bas}`, et une
combinaison `\lohi{en bas}{en haut}`

Ceci est un texte `en haut`, un autre `en bas`, et une
combinaison `en haut`
`en bas`

10.2.4 Texte en verbatim

L'expression latine *verbatim* (de *verbum* = *mot* avec le suffixe *atim*), qui pourrait être traduite par « littéral » ou « mot pour mot », est utilisée dans les programmes de traitement de texte comme ConTEXt pour désigner les fragments de texte qui ne doivent pas être transformé ni mis en forme, ils doivent être affichés, tels quels, dans le fichier final. Pour cela, ConTEXt utilise la commande `\type`, destinée aux textes courts qui n'occupent pas plus d'une ligne et l'environnement `typing` destiné aux textes de plus d'une ligne. Ces commandes sont largement utilisées dans les livres d'informatique pour afficher les fragments de code, et ConTEXt met en forme ces textes en lettres monospace comme le ferait une machine à écrire ou un terminal d'ordinateur. Dans les deux cas, le texte est envoyé dans le document final sans *traitement*, ce qui signifie qu'il peut utiliser des caractères réservés ou des caractères spéciaux qui seront transcrits *tel quel* dans le fichier final. De même, si l'argument de `\type`, ou le contenu de `\starttyping` comprend une commande, celle-ci sera *écrite* dans le document final, mais pas exécutée.

La commande `\type` présente, en outre, la particularité suivante : son argument *peut* être contenu entre des crochets (comme c'est normal dans ConTEXt), mais tout autre caractère peut être utilisé pour délimiter (entourer) l'argument.

Lorsque ConTEXt lit la commande `\type`, il suppose que le caractère qui n'est pas un espace blanc suivant immédiatement le nom de la commande agira comme un délimiteur de son argument ; il considère donc que le contenu de l'argument commence par le caractère suivant, et se termine par le caractère précédent la prochaine apparition du *délimiteur*.

Quelques exemples nous aideront à mieux comprendre :

```
\type 1Tweedledum and Tweedledee A1 \\  
\type pTweedledum and Tweedledee Bp \\  
\type zTweedledum and Tweedledee Cz \\  
\type (Tweedledum and Tweedledee D( \\
```

Tweedledum and Tweedledee A
Tweedledum and Tweedledee B
Tweedledum and Tweedledee C
Tweedledum and Tweedledee D

Notez que dans le premier exemple, le premier caractère après le nom de la commande est un « 1 », dans le deuxième un « | » et dans le troisième un « z » ; ainsi : dans chacun de ces cas, ConTEXt considérera que l'argument de `\type` est tout ce qui se trouve entre ce caractère et la prochaine apparition du même caractère. Il en va de même pour le dernier exemple, qui est également très instructif, car en principe, on pourrait supposer que si le délimiteur d'ouverture de l'argument est un « (», celui de fermeture devrait être un «) », mais ce n'est pas le cas, car « (» et «) » sont des caractères différents et `\type`, comme je l'ai dit, recherche un délimiteur de fermeture identique au caractère utilisé au début de l'argument.

Il n'y a que deux cas où `\type` permet que les délimiteurs d'ouverture et de fermeture soient des caractères différents :

- Si le délimiteur d'ouverture est le caractère « { », il pense que le délimiteur de fermeture sera « } ».
- Si le délimiteur d'ouverture est le caractère « << », il pense que le délimiteur de fermeture sera « >> ». Ce cas est également unique dans la mesure où deux caractères consécutifs sont utilisés comme délimiteurs.

Toutefois, le fait que `\type` autorise n'importe quel délimiteur ne signifie pas que nous devrions utiliser des délimiteurs « bizarre ». Du point de vue de la lisibilité et de l'intelligibilité du fichier source, il est préférable de délimiter l'argument du `\type` par des accolades lorsque cela est possible, comme c'est le cas avec ConTEXt ; et lorsque cela n'est pas possible, parce qu'il y a des accolades dans l'argument du `\type`, utilisez un symbole : de préférence un qui ne soit pas un caractère réservé ConTEXtPar exemple : `\type *Ceci est une accolade fermante : '}'*`.

`\type` et `\starttyping` peuvent être configurés avec `\setuptype` et `\setuptyping`. Nous pouvons également créer une version personnalisée de ces éléments avec `\definetype` et `\definotyping`. En ce qui concerne les options de configuration réelles de ces commandes, je me réfère à « `setup-fr.pdf` » (dans le répertoire `tex/texmf-context/doc/context/documents/general/qrcs`).

Deux commandes très similaires à `\type` sont :

- `\typ` : fonctionne de manière similaire à `\type`, mais ne désactive pas les césures.
- `\tex` : commande destinée à la rédaction de textes sur TeX ou ConTEXt : elle ajoute une barre oblique inversée (backslash ou antislash) devant le texte qu'elle prend comme argument. Sinon, cette commande diffère de `\type` en ce qu'elle traite certains des caractères réservés qu'elle trouve dans le texte qu'elle prend comme argument. En particulier, les crochets à l'intérieur de `\tex` seront traités de la même manière qu'ils le sont habituellement dans ConTEXt.

10.3 Espacement des caractères et des mots

10.3.1 Réglage automatique de l'espacement horizontal

L'espace entre les différents caractères et mots (appelé *espace horizontal* dans TeX) est défini automatiquement par ConTeXt :

- L'espace entre les caractères qui composent un mot est défini par la police elle-même, qui, sauf dans les polices à largeur fixe, utilise généralement une quantité plus ou moins grande d'espace blanc en fonction des caractères à séparer, et donc, par exemple, l'espace entre un « A » et un « V » (« AV ») est généralement moins grand que l'espace entre un « A » et un « X » (« AX »). Cependant, en dehors de ces variations possibles qui dépendent de la combinaison de lettres concernées et prédefinies par la police de caractères, l'espace entre les caractères qui composent un mot est, en général, une mesure fixe et invariable (que ConTeXt peut parfois adapter, mécanisme appelé *expansion* et configuré avec `\definemainfontfeature` et `\setupalign`).

```
\switchtobodyfont[50pt]%
AVA \ \
AXA
```

- En revanche, l'espace entre les mots d'une même ligne peut être plus élastique.
 - Dans le cas des mots d'une ligne dont la largeur doit être la même que celle du reste des lignes du paragraphe, la variation de l'espacement entre les mots est l'un des mécanismes que ConTeXt utilise pour obtenir des lignes de largeur égale, comme expliqué plus en détail dans section 11.3. Dans ces cas, ConTeXt établira exactement le même espace horizontal entre tous les mots de la ligne (sauf pour les règles ci-dessous), tout en s'assurant que l'espace entre les mots des différentes lignes du paragraphe est aussi similaire que possible.
 - Cependant, en plus de la nécessité d'étirer ou de rétrécir l'espacement entre les mots afin de justifier les lignes, selon la langue active, ConTeXt prend en considération certaines règles typographiques selon lesquelles, à certains endroits, la tradition typographique associée à cette langue ajoute un espace blanc supplémentaire, comme c'est le cas, par exemple, dans certaines parties de la tradition typographique anglaise, qui ajoute un espace blanc supplémentaire après un point.

Ces espaces blancs supplémentaires fonctionnent pour l'anglais et peut-être pour d'autres langues (bien qu'il soit également vrai que dans de nombreux cas, les éditeurs anglais choisissent aujourd'hui de ne pas avoir d'espace supplémentaire après un point) mais pas toujours pour le français où la tradition typographique est différente. Nous pouvons donc activer temporairement cette fonction avec `\setupspacing[broad]`

¹⁹ Il est typique de la philosophie de ConTeXt d'inclure une commande pour faire quelque chose que la documentation ConTeXt elle-même décrit conseille de faire. Bien que la perfection typographique soit recherchée, l'objectif est également de donner à l'auteur un contrôle absolu sur l'apparence d'un document. La philosophie de ConTeXt est de faire que les auteurs puissent contrôler tous les aspects de leur document, mais de faire cela de manière transparente et intuitive.

et la désactiver avec `\setupspacing[packed]`. Nous pouvons également modifier la configuration par défaut pour le français avec `\setcharacterspacing [french-punctuation]` (et d'ailleurs pour toute autre langue, y compris l'anglais), comme expliqué dans [section 10.5.2](#).

10.3.2 Modification de l'espace entre les caractères et les mots

Modifier l'espace par défaut des caractères qui composent un mot est considéré comme une très mauvaise pratique d'un point de vue typographique, sauf dans les titres et les en-têtes. ConTeXt fournit une commande pour modifier cet espace entre les caractères d'un mot :¹⁹ `\stretched`, dont la syntaxe est la suivante :

```
\stretched[Configuration]{Texte}
```

où *Configuration* permet l'une des options suivantes :

- **factor** : un nombre décimal qui influence la répartition de l'espace additionnel entre les mots et entre les lettres.
- **width** : indique la largeur totale que doit avoir le texte soumis à la commande, de telle sorte que la commande calcule elle-même l'espacement nécessaire pour répartir les caractères dans cet espace. Par défaut, l'espacement horizontal ajouté comblera la longueur disponible sur la ligne pour le texte donné en argument. Si **factor** devient trop important, alors ConTeXt continuera de faire grandir les espaces sans respecter la valeur de largeur **width** indiquée.

D'après mes tests, lorsque la largeur établie avec l'option **width** est inférieure à celle requise pour représenter le texte avec un **factor** égal à 0.25, l'option **width** et ce facteur sont ignorés. Je suppose que c'est parce que `\stretched` ne nous permet que d'augmenter l'espace entre les caractères d'un mot, et non de le réduire. Mais je ne comprends pas pourquoi la largeur requise pour représenter le texte avec un facteur de 0,25 est utilisée comme mesure minimale pour l'option **width**, et non la *natural width* du texte (avec un facteur de 0).

- **style** : la ou les commandes de style à appliquer au texte pris en argument.
- **color** : couleur dans laquelle le texte pris comme argument sera écrit.

Ainsi, dans l'exemple suivant, nous pouvons voir graphiquement comment la commande fonctionnerait lorsqu'elle est appliquée à la même phrase, mais avec des largeurs différentes :

```
%\setupcharacterkerning
% [stretched]
% [factor=max,
%   width=availablehsize] {\bf to the limit} \\
\stretched[factor=0.00, width=10cm] {\bf to the limit} \\
\stretched[factor=0.25, width=10cm] {\bf to the limit} \\
\stretched[factor=0.50, width=10cm] {\bf to the limit} \\
\stretched[factor=0.75, width=10cm] {\bf to the limit} \\
\stretched[factor=1.00, width=10cm] {\bf to the limit} \\
\stretched[factor=1.25, width=10cm] {\bf to the limit} \\
\stretched[factor=1.50, width=10cm] {\bf to the limit} \\
\stretched[factor=1.75, width=10cm] {\bf to the limit} \\
\\
\stretched[factor=max, width=10cm] {\bf to the limit} \\
\stretched[factor=0.75, width=10cm] {\bf to the limit} \\
\stretched[factor=0.80, width=10cm] {\bf to the limit} \\
\stretched[factor=0.85, width=10cm] {\bf to the limit} \\
\\
\stretched[factor=0.75] {\bf to the limit}
```

Dans cet exemple, en l'absence de `factor`, on peut voir que la répartition de l'espace horizontal entre les différents caractères n'est pas uniforme. Les « T » et « e » dans « Texte » apparaissent toujours beaucoup plus proches les uns des autres que les autres caractères. Je n'ai pas réussi à trouver la raison de ce phénomène.

Appliquée sans argument, la commande utilisera toute la largeur de la ligne. Par contre, dans le texte qui est l'argument de cette commande, la commande `\&` est redéfinie et au lieu d'un saut de ligne, elle insère un espace horizontal. Par exemple :

```
\stretched[width=8cm,factor=0.25] {Texte de test G}\\\n\stretched[width=8cm,factor=0.25] {Texte de \\ test G}
```

Texte	d e	test	G
Texte	d e	test	G

Nous pouvons personnaliser la configuration par défaut de la commande avec `\setupstretched`.



Il n'existe pas de commande `\definestretched` qui nous permettrait de définir des configurations personnalisées associées à un nom de commande, cependant, dans la liste officielle des commandes (voir section ??), il est indiqué que `\setupstretched` provient de `\setup-characterkerning` et qu'il existe une commande `\definecharacterkerning`. Dans mes tests, cependant, je n'ai pas réussi à définir une configuration personnalisée pour `\stretched` au moyen de cette dernière, bien que je doive admettre que je n'ai pas passé beaucoup de temps à essayer de le faire non plus.

10.3.3 Commandes pour ajouter un espace horizontal entre les mots

Nous savons déjà que pour augmenter l'espace entre les mots, il est inutile d'ajouter deux espaces vides consécutifs ou plus, car ConTEXt absorbe tous les espaces vides consécutifs, comme expliqué dans [section 4.2.1](#). Si nous souhaitons augmenter l'espace entre les mots, nous devons passer par l'une des commandes qui nous permet de le faire :

- `\,`, insère un très petit espace vide (appelé espace mince) dans le document. Il est utilisé, par exemple, pour séparer les milliers dans un ensemble de chiffres (par exemple 1,000,000), ou pour séparer une simple virgule inversée des doubles virgules inversées. Par exemple :

```
1 473 451 \\\n1\,473\,451 \\\n
```

```
1 473 451\n1 473 451
```

- `\space` ou « `_` » (une barre oblique inversée suivie d'un espace vide que j'ai représenté par « `_` », car il s'agit d'un caractère invisible) introduit un espace vide supplémentaire.
- `\enskip`, `\quad` et `\qquad` insèrent un espace blanc dans le document de respectivement un demi-*em*, *1em* ou *2ems*. N'oubliez pas que le *em* est une mesure qui dépend de la taille de la police et équivaut à la largeur d'un « `m` », qui coïncide normalement avec la taille en points de la police. Ainsi, en utilisant une police de 12 points, `\enskip` nous donne un espace de 6 points, `\quad` nous donne 12 points et `\qquad` nous donne 24 points.

Outre ces commandes qui nous donnent des espaces vides de dimensions précises, les commandes `\hskip` et `\hfill` introduisent des espaces horizontaux de dimensions variables :

\hspace nous permet d'indiquer exactement la quantité d'espace vide que nous voulons ajouter. L'espace indiqué peut être négatif, ce qui entraînera la superposition d'un texte sur un autre. Ainsi :

```
Ceci est un espace de \hspace{1.0cm} 1 centimetre\\
Ceci est un espace de \hspace{2.0cm} 2 centimètres\\
Ceci est un espace de \hspace{2.5cm} 2.5 centimètres\\
Ceci est un espace de \hspace{-1.0cm} moins 1 centimetre\\
```

```
Ceci est un espace de 1 centimetre
Ceci est un espace de 2 centimètres
Ceci est un espace de 2.5 centimètres
Ceci est un espace de moins 1 centimetre
```

\hfill, pour sa part, introduit autant d'espace blanc que nécessaire pour occuper toute la ligne, ce qui nous permet de créer des effets intéressants tels que du texte aligné à droite, du texte centré ou du texte des deux côtés de la ligne. Dans le cas de plusieurs \hfill sur une ligne, ils se répartissent l'espace horizontal de façon équitable. Par exemple :

```
\hfill hfill pousse à droite.\\
hfill pousse \hfill des deux côtés.\\
2 hfills \hfill poussent \hfill simultanément.
```

hfill pousse
2 hfills poussent

hfill pousse à droite.
des deux côtés.
simultanément.

Une dernière commande, \wordright (également détaillée en section 11.6.1), propose un fonctionnement similaire à \hfill au sens où elle va pousser le texte qu'elle prend en argument vers la droite, mais elle va en plus automatiquement passer à la ligne, juste après son argument. Voyez :

```
hfill pousse \hfill des deux côtés. 2 hfills
\hfill poussent \hfill simultanément.

hfill pousse \wordright{des deux côtés.} 2
hfills \hfill poussent \hfill simultanément.
```

hfill pousse des deux côtés. 2 hfills poussent
simultanément.
hfill pousse
2 hfills poussent
des deux côtés.
simultanément.

10.4 Mots composés

Par « mots composés » dans cette section, j'entends les mots qui sont formellement compris comme étant un seul mot, plutôt que les mots qui sont simplement conjoints. Cette distinction n'est pas toujours facile à comprendre : « portemanteau » est clairement composé de deux mots (« porte + manteau ») mais aucun francophone ne penserait aux termes combinés d'une autre manière que comme un seul mot. D'autre part, nous avons des mots qui sont parfois combinés à l'aide d'un trait d'union ou d'une barre oblique inversée. Les deux mots ont des significations et des utilisations distinctes, mais ils sont joints (et peuvent dans certains cas devenir un seul mot, mais pas encore !) Ainsi, par exemple, nous pouvons trouver des mots comme « Français–Canadien » ou « (inter)communication » (bien que nous puissions aussi trouver « intercommunication » et découvrir que l'usage public a finalement accepté que les deux mots soient un seul mot. C'est ainsi que les langues évoluent).

Les mots composés posent quelques problèmes, principalement liés à leur éventuelle césure en fin de ligne. Si l'élément de jonction est un trait d'union, d'un point de vue typographique, il n'y a pas de problème de césure à la fin d'une ligne à cet endroit, mais nous devrions éviter une deuxième césure dans la deuxième partie du mot, car cela nous laisserait avec deux traits d'union consécutifs qui pourraient causer des difficultés de compréhension.

La commande `||` est disponible pour indiquer à ConTeXt que deux mots forment un mot composé. Cette commande, exceptionnellement, ne commence pas par une barre oblique inverse, et permet deux utilisations différentes :

- Nous pouvons utiliser deux barres verticales consécutives (tube ou pipe en anglais) et écrire, par exemple, `Espagnol||Argentin`.
- Les deux barres verticales peuvent être séparées de l'élément de jonction à utiliser entre les deux mots, comme dans, par exemple, `joindre||/séparer`.

Dans les deux cas, ConTeXt saura qu'il s'agit d'un mot composé et appliquera les règles de césure appropriées pour ce type de mot. La différence entre l'utilisation des deux barres verticales consécutives (tubes), ou l'encadrement du séparateur de mot avec celles-ci, est que dans le premier cas, ConTeXt utilise le séparateur prédéfini comme `\setuphyphenmark`, ou en d'autres termes le trait d'union, qui est la valeur par défaut (« -- »). Ainsi, si nous écrivons « `après||midi` », ConTeXt générera « `après-midi` ». alors qu'avec « `après||/mid`i »), ConTeXt générera « `après/midi` ».

Avec `\setuphyphenmark`, nous pouvons changer le séparateur par défaut (dans le cas où nous utilisons les deux barres verticales). Les valeurs autorisées pour cette commande sont « --, ---, -, , (,), =, / ». N'oubliez pas, cependant, que la valeur « = » devient un tiret (comme « --- »).

L'utilisation normale de « `||` » est avec des tirets, puisque c'est ce qui est normalement utilisé entre les mots composés. Mais parfois, le séparateur peut être une parenthèse, si, par exemple, nous voulons obtenir « (inter)space », ou il peut être une barre oblique, comme dans « input/output ». Dans ces cas, si nous voulons que

les règles de césure normales pour les mots composés s'appliquent, nous pouvons écrire « (inter|) | space » ou « input | / | output ». Comme je l'ai dit précédemment, « |=| » est considéré comme une abréviation de « |---| » et insère un tiret em comme séparateur (—).

10.5 La langue du texte

²⁰ Table 10.5 a un résumé de la liste obtenue avec les commandes suivantes :
`\usemodule[languages-system]
\loadinstalledlanguages
\showinstalledlanguages`

Si vous lisez ce document longtemps après qu'il ait été écrit (2020), il est possible que ConTeXt ait incorporé d'autres langues, il serait donc judicieux d'utiliser ces commandes pour afficher une liste mise à jour des langues.

Les caractères forment des mots qui appartiennent normalement à une certaine langue. Il est important pour ConTeXt de connaître la langue dans laquelle nous écrivons, car un certain nombre de choses importantes en dépendent. Principalement :

- La césure des mots.
- Le format de sortie de certains mots.
- Certaines questions de composition associées à la tradition de composition de la langue en question.

10.5.1 Définition et modification de la langue

Par défaut, ConTeXt suppose que la langue sera l'anglais. Deux procédures permettent de changer cela :

- En utilisant la commande `\mainlanguage`, utilisée en préambule pour changer la langue principale du document. Cela permet d'adapter les éléments générés automatiquement par ConTeXt. « Table of content » et « Chapter », deviennent respectivement « Table des matières » ry « Chapitre ».
- En utilisant la commande `\language`, visant à changer la langue active à tout moment du document. Cette commande modifie les règles de césure, les guillemets, etc.

Les deux commandes attendent un argument consistant en un identifiant (ou code) de langue quelconque. Pour identifier la langue, nous utilisons soit le code international de langue à deux lettres défini dans la norme ISO 639-1, qui est le même que celui utilisé, par exemple, sur le web, soit le nom anglais de la langue en question, ou parfois une abréviation du nom en anglais.

Dans la [table 10.5²⁰](#), nous trouvons une liste complète des langues supportées par ConTeXt, avec le code ISO pour chacune des langues en question ainsi que, le cas échéant, le code pour certaines variantes linguistiques expressément prévues.

Ainsi, par exemple, pour définir le français comme langue principale du document, nous pouvons utiliser l'une des trois codes suivants :

```
\mainlanguage[fr]
\mainlanguage[fra]
\mainlanguage[french]
```

Pour activer une langue particulière *dans* le document, nous pouvons utiliser soit la commande `\language[Code de la langue]`, soit une commande spécifique pour activer cette langue. Ainsi, par exemple, `\en` active l'anglais, `\fr` active le français, `\es` espagnol, ou `\ca` le catalan. Une fois qu'une langue a été activée, elle le reste jusqu'à ce que l'on passe expressément à une autre langue, ou que le groupe dans lequel la langue a été activée soit alors fermé. Les langues fonctionnent donc

Langage	Code ISO	Langage (variantes)
Afrikaans	af, afrikaans	
Arabic	ar, arabic	ar-ae, ar-bh, ar-dz, ar-eg, ar-in, ar-ir, ar-jo, ar-kw, ar-lb, ar-ly, ar-ma, ar-om, ar-qa, ar-sa, ar-sd, ar-sy, ar-tn, ar-ye
Catalan	ca, catalan	
Czech	cs, cz, czech	
Croatian	hr, croatian	
Danish	da, danish	
Dutch	nl, nld, dutch	
English	en, eng, english	en-gb, uk, ukenglish, en-us, usenglish
Estonian	et, estonian	
Finnish	fi, finnish	
French	fr, fra, french	
German	de, deu, german	de-at, de-ch, de-de
Greek	gr, greek	
Greek (ancient)	agr, ancientgreek	
Hebrew	he, hebrew	
Hungarian	hu, hungarian	
Italian	it, italian	
Japanese	ja, japanese	
Korean	kr, korean	
Latin	la, latin	
Lithuanian	lt, lithuanian	
Malayalam	ml, malayalam	
Norwegian	nb, bokmal, no, nor-	nn, nynorsk wegian
Persian	pe, fa, persian	
Polish	pl, polish	
Portuguese	pt, portuguese	pt-br
Romanian	ro, romanian	
Russian	ru, russian	
Slovak	sk, slovak	
Slovenian	sl, slovene, slovenian	
Spanish	es, sp, spanish	es-es, es-la
Swedish	sv, swedish	
Thai	th, thai	
Turkish	tr, turkish	tk, turkmen
Ukrainian	ua, ukrainian	
Vietnamese	vi, vietnamese	

Tableau 10.5 Support des langues avec ConTeXt

comme des commandes de changement de police. Notez toutefois que la langue définie par la commande `\language` ou par l'une de ses abréviations (`\en`, `\fr`, `\de`, etc.) n'affecte pas la langue dans laquelle les étiquettes sont imprimées (voir section 10.5.3).

Bien qu'il puisse être laborieux de marquer la langue de tous les mots et expressions que nous utilisons dans notre document et qui n'appartiennent pas à la langue principale du document, il est important de le faire si nous voulons obtenir un document final correctement composé, en particulier dans les travaux professionnels. Nous ne devons pas marquer tout le texte, mais seulement la partie qui n'appartient pas à la langue principale. Il est parfois possible d'automatiser le marquage de la langue en utilisant une macro. Par exemple, pour ce document dans lequel on cite continuellement des commandes ConTeXt dont la langue d'origine est l'anglais, j'ai conçu une macro qui, en plus d'écrire la commande dans le format et la couleur appropriés, la marque comme un mot anglais. Dans mon travail professionnel, où je dois citer beaucoup de bibliographie française et italienne, j'ai incorporé un champ dans ma base de données bibliographique pour capter la langue de l'ouvrage, de façon à automatiser l'indication de la langue dans les citations et les listes de références bibliographiques.

Si nous utilisons deux langues qui utilisent des alphabets différents dans le même document (par exemple, l'anglais et le grec, ou l'anglais et le russe), il existe une astuce qui nous évitera de devoir marquer la langue des expressions construites avec l'alphabet alternatif : modifier le paramètre de la langue principale (voir section suivante) afin qu'il charge également les modèles de césure par défaut pour la langue qui utilise un alphabet différent. Par exemple, si nous voulons utiliser l'anglais et le grec ancien, la commande suivante nous évitera de devoir marquer la langue des textes en grec :

```
\setuplanguage [en] [patterns={en, agr}]
```

Cela ne fonctionne que parce que l'anglais et le grec utilisent un alphabet différent. Il ne peut donc y avoir de conflit entre les modèles de césure des deux langues, et nous pouvons donc les charger simultanément. Mais dans deux langues qui utilisent le même alphabet, le chargement simultané des modèles de césure conduira nécessairement à une césure inappropriée.

10.5.2 Configuration de la langue

ConTeXt associe le fonctionnement de certains utilitaires à la langue spécifique active à un moment donné. Les associations par défaut peuvent être modifiées avec `\setuplanguage` dont la syntaxe est :

```
\setuplanguage [Langage] [Configuration]
```

où *Langage* est le code de la langue que nous voulons configurer, et *Configuration* contient la configuration spécifique que nous voulons définir (ou modifier) pour cette langue. Plus précisément, jusqu'à 32 options de configuration différentes sont autorisées, mais je ne traiterai que celles qui semblent convenir à un texte d'introduction tel que celui-ci :

- **date** : permet de configurer le format de date par défaut. Voir plus loin sur [page 261](#).
- **lefthyphenmin**, **righthyphenmin** : le nombre minimum de caractères qui doivent se trouver à gauche ou à droite pour que la césure d'un mot soit prise en charge. Par exemple, `\setuplanguage [en] [lefthyphenmin=4]` ne prendra pas en charge la césure d'un mot s'il y a moins de 4 caractères à gauche du trait d'union éventuel.

- **spacing** : les valeurs possibles pour cette option sont « *broad* » ou « *packed* ». Dans le premier cas (*broad*), les règles d'espacement des mots en anglais seront appliquées, ce qui signifie qu'après un point et lorsqu'un autre caractère suit, une certaine quantité d'espace vide supplémentaire sera ajoutée. En revanche, « *spacing=packed* » empêchera l'application de ces règles. Pour l'anglais, *broad* est la valeur par défaut.
- **leftquote**, **rightquote** : indique les caractères (ou commandes), respectivement, que `\quote` utilisera à gauche et à droite du texte qui est son argument (pour cette commande, voir [page 263](#)).
- **leftquotation**, **rightquotation** : indique les caractères (ou commandes), respectivement que `\quotation` utilisera à gauche et à droite du texte qui est son argument (pour cette commande, voir [page 263](#)).

Voyez par exemple l'effet des configurations par défaut :

```
\language[en] \quotation{Mon texte} et \quote{Mon texte} \\  
\language[de] \quotation{Mon texte} et \quote{Mon texte} \\  
\language[fr] \quotation{Mon texte} et \quote{Mon texte}
```

"Mon texte" et 'Mon texte'
„Mon texte“ et ,Mon texte'
« Mon texte » et «Mon texte»

Dans le cas particulier du français, n'oubliez pas d'utiliser `\setcharacterspacing[frenchpunctuation]` qui indique à ConTeXt d'ajouter des espaces fines insécables devant les signes doubles (mais pas après, sauf pour « »). Il n'enlève pas les espaces en trop donc il vous faudra tout de même faire attention au niveau du fichier source. Vous pouvez aussi configurer directement ce comportement avec `\setupcharacterspacing`.

```
\language[fr]%
A - Une phrase,avec des,signes simples.\\
B - Une:phrase;avec des signes! doubles?\\
C - Une phrase , avec des , signes simples .\\
D - Une : phrase ; avec des signes ! doubles ?\\
E - et\quote{Mon texte}pour finir.\\

\blank[big]
\setcharacterspacing [frenchpunctuation]

A - Une phrase,avec des,signes simples.\\
B - Une:phrase;avec des signes! doubles?\\
C - Une phrase , avec des , signes simples .\\
D - Une : phrase ; avec des signes ! doubles ?\\
E - et\quote{Mon texte}pour finir.\\

\blank[big]
\setupcharacterspacing [frenchpunctuation] ["003A] % pour :
[left=.75,right=.75,alternative=1]
% ["003A]: ["003B]; ["003F]? ["0021]! ["002C], ["002E]. ["00AB]« ["00BB]»
B - Une:phrase;avec des signes! doubles?
```

- A - Une phrase,avec des,signes simples.
 B - Une:phrase;avec des signes! doubles?
 C - Une phrase , avec des , signes simples .
 D - Une : phrase ; avec des signes ! doubles ?
 E - et«Mon texte»pour finir.
- A - Une phrase,avec des,signes simples.
 B - Une :phrase ;avec des signes ! doubles ?
 C - Une phrase , avec des , signes simples .
 D - Une : phrase ; avec des signes ! doubles ?
 E - et« Mon texte »pour finir.
- B - Une : phrase ;avec des signes ! doubles ?

10.5.3 Étiquettes associées à des langues particulières

De nombreuses commandes de ConTeXt génèrent automatiquement certains textes (ou *étiquettes*), comme, par exemple, la commande `\placetable` qui écrit l'étiquette « Tableau xx » sous le tableau qui est inséré, ou `\placefigure` qui insère l'étiquette « Figure xx ».

Ces étiquettes dépendent de la langue définie avec `\mainlanguage`. (mais pas de `\language` qui ne concerne que les règles de césure, ponctuations, guillements etc.) et nous pouvons les modifier avec la commande :

```
\setuplabeltext [Language] [Clé=Texte]
```

où *Clé* est le terme par lequel ConTeXt connaît l'étiquette et *Texte* est le texte que nous voulons que ConTeXt génère. Ainsi, par exemple,

```
\mainlanguage[fr]%
\setuplabeltext[fr][figure={Illustration~}]
\placefigure[][]
{Texte de la légende}
{Texte}
```

Texte

Illustration 1 Texte de la légende

implique que lorsque la langue principale est le français, les images insérées avec `\placefigure` ne sont pas appelées « Figure x » mais « Illustration x ». Notez qu'après le texte de l'étiquette proprement dite, il faut laisser un espace vide pour que l'étiquette ne soit pas rattachée au caractère suivant qui est la numérotation de la figure. Dans l'exemple, j'ai utilisé le caractère réservé « ~ » ; j'aurais également pu écrire « [figure=Illustration{ }] » en enfermant l'espace vide entre des accolades pour m'assurer que ConTeXt ne s'en débarrasse pas.

Quelles étiquettes pouvons-nous redéfinir avec `\setuplabeltext` ? La documentation de ConTeXt n'est pas aussi complète qu'on pourrait l'espérer sur ce point. Le manuel de référence 2013 (qui est celui qui explique le plus cette commande) mentionne « chapter », « table », « figure », « appendix »... et ajoute « autres éléments de texte comparables ». Nous pouvons notamment identifier « content », « tables », « figures », « index », « intermezzo », « intermezzi », « graphic », « graphics », « abbreviations », « logos », « units », « part », « section », « subsection », « subsubsection », « subsubsubsection », « line », « lines », « page », « atpage », « henceforth », « hereafter », « see ».

L'un des avantages des *logiciels libres* est la disponibilité des fichiers sources ; nous pouvons donc les examiner. C'est ce que j'ai fait, et en fouinant dans les fichiers sources de ConTeXt, j'ai découvert le fichier `lang-txt.lua`, disponible dans `tex/texmf-context/tex/context/base/mkiv` qui, je pense, est celui qui contient les étiquettes prédefinies et leurs différentes traductions ; de sorte que si à tout moment ConTeXt génère un texte redéfini que nous voulons changer, pour identifier le nom de l'étiquette à laquelle ce texte est associé il suffit de consulter ce fichier.

Si nous voulons insérer le texte associé à une certaine étiquette quelque part dans le document, nous pouvons le faire avec la commande `\labeltext`. Ainsi, par exemple, si je veux faire référence à un tableau, pour m'assurer que je le nomme de la même manière que ConTeXt dans la commande `\placetable`, je peux écrire : « Juste comme indiqué dans le `\labeltext{table}` de la page suivante. ».

```
\useMPlibrary[dum] % pour produire des images
\mainlanguage[fr]%
\setuplabeltext[fr][figure={Illustration~}]
\placefigure[][]
{Texte de la légende}
{\externalfigure[dummy]
[height=1cm,width=6cm]}
```

Juste comme indiqué dans l'`\labeltext{figure}` précédente.

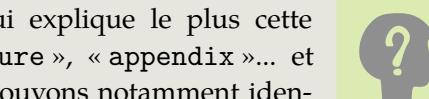


Illustration 1 Texte de la légende

Juste comme indiqué dans l'Illustration précédente.

Certaines étiquettes redéfinissables avec `\setuplabeltext`, sont vides par défaut ; comme, par exemple, « chapter » ou « section ». Cela est dû au fait que, par défaut, ConTeXt n'ajoute pas d'étiquettes aux commandes de sectionnement. Si nous voulons changer ce fonctionnement par défaut, il nous suffit de redéfinir ces étiquettes dans le préambule de notre document et ainsi, par exemple, `\setuplabeltext[chapter=Chapter~]` verra que les chapitres sont précédés du mot « Chapter ».

```
\mainlanguage[fr]%
\setuplabeltext[fr][section={Verset~}]%

\startsection[title=Titre de section]
\stopsection
```

Verset 1 Titre de section

Enfin, il est important de souligner que bien qu'en général, dans ConTeXt, les commandes qui permettent plusieurs options séparées par des virgules comme argument, la dernière option peut se terminer par une virgule et rien de mal ne se passe. Dans `\setuplabeltext`, cela générera une erreur lors de la compilation.

10.5.4 Quelques commandes liées à la langue

A. Commandes liées à la date

ConTeXt a trois commandes liées à la date qui produisent une sortie dépendante de la langue active au moment où elles sont exécutées. Il s'agit de :

- `\currentdate` : exécutée sans arguments dans un document dont la langue principale est l'anglais, elle renvoie la date système au format « Jour Mois Année ». Par exemple « 11 septembre 2020 ». Mais nous pouvons également lui demander d'utiliser un format différent (comme c'est le cas aux États-Unis et dans certaines autres parties du monde anglophone qui suivent leur système consistant à placer le mois avant le jour, d'où la date tristement célèbre du 11 septembre), ou d'inclure le nom du jour de la semaine (`weekday`), ou de n'inclure que certains éléments de la date (`day`, `month`, `year`).

Pour indiquer un format de date différent, « `dd` » ou « `day` » représentent les jours, « `mm` » les mois (au format numérique), « `month` » les mois au format alphabétique en minuscules, et « `MONTH` » en majuscules. En ce qui concerne l'année, « `yy` » n'écrira que les derniers chiffres, tandis que « `year` » ou « `y` » écrira les quatre. Si nous voulons un élément de séparation entre les composants de la date, nous devons l'écrire expressément. Par exemple

```
\mainlanguage[en] \language[en]%
\currentdate[weekday,month,day,{,},year]

\mainlanguage[fr] \language[fr]%
\currentdate[weekday,day,month,year]
```

Sunday June 27, 2021
dimanche 27 juin 2021

- `\date` : cette commande, exécutée sans aucun argument, produit exactement le même résultat que `\currentdate`, c'est-à-dire la date actuelle au format standard. Cependant, une date spécifique peut être donnée comme argument. Deux arguments sont fournis pour cela : avec le premier argument, nous pouvons indiquer le jour (« `d` »), le mois (« `m` ») et l'année (« `y` ») correspondant à la date

que nous voulons représenter, tandis qu'avec le second argument (facultatif), nous pouvons indiquer le format de la date à représenter. Par exemple, si nous voulons savoir quel jour de la semaine John Lennon et Paul McCartney se sont rencontrés, un événement qui, selon Wikipedia, a eu lieu le 6 juillet 1957, nous pourrions écrire

```
\mainlanguage[fr] \language[fr]%
\date[d=6, m=7, y=1957] [weekday]
```

samedi

et ainsi nous découvrons qu'un tel événement historique s'est produit un samedi.

- `\month` prend un nombre en argument, et renvoie le nom du mois correspondant à ce nombre.

Pour l'heure, il existe aussi `\currenttime` qui prend en argument « h » pour les heures et « m » pour les minutes.

B. La commande `\translate`

La commande `\translate` prend en charge une série de phrases associées à une langue spécifique, de sorte que l'une ou l'autre sera insérée dans le document final en fonction de la langue active à un moment donné. Dans l'exemple suivant, la commande `translate` est utilisée pour associer quatre phrases au français et à l'anglais, qui sont enregistrées dans un tampon de mémoire (concernant l'environnement `buffer`, voir [section 12.6](#)). Si nous insérons le *tampon* à un point du document où le français est activé, les phrases en français seront affichées, et de même avec l'anglais. Ainsi :

```
\startbuffer
\starttabulate[|*{4}{1w(3cm)}|]
\HL \NC \translate[fr=Votre lettre, en=Your letter]
    \NC \translate[fr=Votre référence, en=Your reference]
    \NC \translate[fr=Notre référence, en=Our reference]
    \NC \translate[fr=Date, en=Date] \NR
\stoptabulate
\stopbuffer

\language[fr] \getbuffer
\blank[big]
\language[en] \getbuffer
```

Votre lettre	Votre référence	Notre référence	Date
Your letter	Your reference	Our reference	Date

C. Les commandes `\quote` et `\quotation`

L'une des erreurs typographiques les plus courantes dans les documents texte survient lorsque les guillemets (simples ou doubles) sont ouverts mais pas expressément fermés. Pour éviter cela, ConTeXt fournit les commandes `\quote` et `\quotation` qui citeront le texte qui est leur argument ; `\quote` utilisera des guillemets simples et `\quotation` des guillemets doubles.

Ces commandes sont sensibles à la langue dans la mesure où elles utilisent le jeu de caractères ou de commandes par défaut de la langue en question pour ouvrir et fermer les guillemets (cf. section 10.5.2) ; Dans l'exemple suivant, vous voyez la configuration pour le français par défaut puis une version utilisant les versions anglaises des guillemets simples et doubles.

```
\setcharacterspacing [frenchpunctuation]%
\mainlanguage[fr] \language[fr]%
\tdf \quotation{Mon texte} et \quote{Mon texte}
\blank[big]
\setuplanguage[fr] [leftquotation=",rightquotation="]
\setuplanguage[en] [leftquote=' ,rightquote=']
\tdf \quotation{Mon texte} et \quote{Mon texte}
```

« Mon texte » et « Mon texte »
“ Mon texte ” et « Mon texte »

Ces commandes ne gèrent cependant pas les guillemets imbriqués, bien que nous puissions créer un utilitaire qui le fasse, en profitant du fait que `\quote` et `\quotation` sont des applications réelles de ce que ConTeXt appelle *delimitedtext*, et qu'il est possible de définir d'autres applications avec `\definedelimitedtext`. Ainsi, l'exemple suivant :

crée trois commandes qui permettront jusqu'à trois niveaux différents de citation. Le premier niveau avec des guillemets latéraux, le deuxième avec des guillemets doubles et le troisième avec des guillemets simples.

Bien entendu, si nous utilisons l'anglais comme langue principale, les guillemets simples et doubles par défaut (frisés, et non droits, comme dans ce document !) seront automatiquement utilisés.

```
\definedelimitedtext [GuillemetsNivA] [left=<, right=>]  
\definedelimitedtext [GuillemetsNivB] [left=", right="]  
\definedelimitedtext [GuillemetsNivC] [left=' , right=']  
  
\GuillemetsNivA{Mon texte et \GuillemetsNivB{un second et \GuillemetsNivC{un troisième}}}
```

«Mon texte et “un second et ‘un troisième”»

Enfin, sachez qu'il existe un environnement spécifique pour présenter les citation :
`\startquotation` présenté page 340.

Chapitre 11

Paragraphes, lignes et espace vertical

Table of Contents : [11.1 Les paragraphes et leurs caractéristiques](#) ; [11.1.1 Mise en retrait automatique des premières lignes des paragraphes](#) ; [11.1.2 Mise en retrait de paragraphes](#) ; [11.2 Espace vertical entre les paragraphes](#) ; [11.2.1 \setupwhitespace](#) ; [11.2.2 Paragraphes sans espace vertical supplémentaire entre eux](#) ; [11.2.3 Ajout d'un espace vertical supplémentaire à un endroit particulier du document](#) ; [11.2.4 \setupblank et \defineblank](#) ; [11.2.5 Autres procédures pour obtenir plus d'espace vertical](#) ; [11.3 Comment ConTeXt construit les lignes qui forment les paragraphes](#) ; [11.3.1 Utilisation du caractère réservé « ~ »](#) ; [11.3.2 La césure des mots](#) ; [11.3.3 Niveau de tolérance pour les sauts de ligne](#) ; [11.3.4 Forcer un saut de ligne à un certain point](#) ; [11.4 Espace interligne](#) ; [11.5 Autres questions relatives aux lignes](#) ; [11.5.1 Conversion des sauts de ligne du fichier source en sauts de ligne du document final](#) ; [11.5.2 Numérotation des lignes](#) ; [11.6 Alignement horizontal et vertical](#) ; [11.6.1 Alignement horizontal](#) ; [11.6.2 Alignement vertical](#) ;

L'aspect général d'un document est principalement déterminé par la taille et la disposition des pages que nous avons vues dans le [Chapitre 5](#), par la police de caractères que nous avons choisie, traitée dans le Chapitre ??, et par d'autres questions comme l'espacement entre les lignes, l'alignement des paragraphes et l'espacement entre eux, etc. Ce chapitre se concentre sur ces questions.

11.1 Les paragraphes et leurs caractéristiques

Le paragraphe est l'unité de texte fondamentale pour ConTeXt. Il existe deux procédures pour commencer un paragraphe :

1. Insertion d'une ou plusieurs lignes vides consécutives dans le fichier source.
2. Les commandes `\par` ou `\endgraf`.

La première de ces procédures est celle qui est normalement utilisée car elle est plus simple et produit des fichiers sources plus faciles à lire et à comprendre. L'insertion de sauts de paragraphe par une commande explicite n'est généralement réalisée qu'à l'intérieur d'une macro (voir [section 3.7.1](#)) ou dans une cellule de tableau (voir [section 13.3](#)).

Dans un document bien typé, il est important, d'un point de vue typographique, que les paragraphes se distinguent visuellement les uns des autres. On y parvient généralement par deux procédés : en indentant légèrement la première ligne de chaque paragraphe ou en augmentant légèrement l'espace blanc entre les paragraphes, et parfois par une combinaison des deux procédés, bien que dans certains endroits, cette méthode ne soit pas recommandée car elle est considérée comme redondante du point de vue typographique.

Je ne suis pas tout à fait d'accord. Le simple retrait de la première ligne ne souligne pas toujours assez visuellement la séparation entre les paragraphes ; mais une augmentation de l'espacement non accompagnée d'un retrait pose des problèmes dans le cas d'un paragraphe qui commence en haut d'une page où nous ne pouvons pas savoir s'il s'agit d'un nouveau paragraphe, ou d'une continuation de la page précédente. Une combinaison des deux procédures permet d'éliminer les doutes.

Voyons tout d'abord comment l'indentation des lignes et des paragraphes est réalisée avec ConTeXt.

11.1.1 Mise en retrait automatique des premières lignes des paragraphes

L'insertion automatique d'un petit retrait à la première ligne des paragraphes est désactivée par défaut. Nous pouvons l'activer, la désactiver à nouveau et, lorsqu'elle est activée, indiquer l'étendue du retrait à l'aide de la commande `\setupindenting` qui autorise les valeurs suivantes pour indiquer si le retrait doit être activé ou non :

- **always** : tous les paragraphes seront indentés, quoi qu'il arrive.
- **yes** : active l'indentation des paragraphes *normaux*. Certains paragraphes précédés d'un espace vertical supplémentaire, comme le premier paragraphe des sections, ou les paragraphes suivant certains environnements, ne seront pas mis en retrait.
- **no, not, never, none** : désactive le retrait automatique de la première ligne des paragraphes.
- **first / next** : active ou désactive le retrait automatique du premier paragraphe d'une section.

Dans le cas où nous avons activé l'indentation automatique, nous pouvons également indiquer, au moyen de la même commande, la quantité d'indentation à prévoir. Pour ce faire, nous pouvons utiliser expressément une dimension (par exemple 1,5 cm) ou bien les mots symboliques « `small` », « `medium` » et « `big` » qui indiquent que nous voulons un petit, moyen ou grand retrait.

```
\setupindenting[yes,1em]
Un petit paragraphe afin de montrer l'effet visuel d'une indentation. Un petit paragraphe afin de montrer l'effet visuel d'une indentation.
```

```
\setupindenting[yes,medium]
Un petit paragraphe afin de montrer l'effet visuel d'une indentation. Un petit paragraphe afin de montrer l'effet visuel d'une indentation.
```

```
\setupindenting[no,2em]
Un petit paragraphe afin de montrer l'effet visuel d'une indentation. Un petit paragraphe afin de montrer l'effet visuel d'une indentation.
```

Un petit paragraphe afin de montrer l'effet visuel d'une indentation. Un petit paragraphe afin de montrer l'effet visuel d'une indentation.

Un petit paragraphe afin de montrer l'effet visuel d'une indentation. Un petit paragraphe afin de montrer l'effet visuel d'une indentation.

Un petit paragraphe afin de montrer l'effet visuel d'une indentation. Un petit paragraphe afin de montrer l'effet visuel d'une indentation.

Dans certaines traditions de composition (et notamment en espagnol), l'indentation par défaut était de deux quarts. En typographie, une quad (à l'origine *quadrat*) était un espaceur (*entretoise*) métallique utilisé en typographie. Le terme a ensuite été adopté comme nom générique pour deux tailles d'espacement courantes en typographie, quelle que soit la forme de composition utilisée. Un em quad est un espace d'un em de large, c'est-à-dire aussi large que la lettre M de la police. Ainsi, dans le cas d'une lettre de 12 points, le quadrilatère aurait une largeur de 12 points et une hauteur de 12 points. ConTeXt possède deux commandes de quadrature : \quad qui génère un espace du type mentionné ci-dessus, et \quadquad qui génère deux fois cette quantité, toujours en restant proportionnel à la police utilisée. Un retrait de deux quads avec une lettre de 11 points mesurera 22 points, et avec une lettre de 12 points, 24 points.

Lorsque l'indentation est activée, si nous ne voulons pas qu'un certain paragraphe soit indenté, nous devons utiliser la commande \noindentation.

En général, j'active l'indentation automatique dans mes documents avec \setupindenting [yes, big]. Dans ce document, cependant, je ne l'ai pas fait parce que si l'indentation était activée, le grand nombre de phrases courtes et d'exemples donnerait une apparence désordonnée aux pages.

11.1.2 Mise en retrait de paragraphes

Une méthode graphique pour mettre en évidence un paragraphe consiste à mettre en retrait le côté droit ou le côté gauche (ou les deux) d'un paragraphe. Cette méthode est utilisée, par exemple, pour les blocs de citation.

ConTeXt propose un environnement dédié pour cela : « narrower ».

```
\startnarrower [Options] ... \stopnarrower
```

où Options peut être :

- **left** : met en retrait la marge de gauche.
- **Num*gauche** : met en retrait la marge de gauche, en multipliant le retrait *normal* par *Num*. (par exemple, $2*\text{left}$).
- **right** : met en retrait la marge de droite.
- **Num*right** : met en retrait la marge de droite, en multipliant le retrait *normal* par *Num*. (par exemple, $2*\text{right}$).
- **middle** : met en retrait les deux marges. Il s'agit de la valeur par défaut.
- **Num*middle** : met en retrait les deux marges, en multipliant le retrait *normal* par *Num*.

Lors de l'explication des options, j'ai mentionné l'indentation *normale*, qui fait référence à la quantité d'indentation gauche et droite que « `narrower` » applique par défaut. Cette quantité peut être configurée avec `\setupnarrower` qui permet les options de configuration suivantes :

- **left** : montant de l'indentation à appliquer à la marge gauche.
- **right** : montant du retrait à appliquer à la marge de droite.
- **middle** : montant du retrait à appliquer aux deux marges.
- **before** : commande à exécuter avant d'entrer dans l'environnement.
- **after** : commande à exécuter après avoir existé dans l'environnement.

Si nous voulons utiliser différentes configurations de l'environnement plus étroit dans notre document, nous pouvons attribuer un nom différent à chacune d'entre elles avec `\definenarrower [Name] [Configuration]` où *Name* est le nom lié à cette configuration et où *Configuration* autorise les mêmes valeurs que `\setupnarrower`.

```
\definenarrower[ConfigA][left=1cm,right=3cm,default={right,left}]
\definenarrower[ConfigB][left=3cm,right=1cm,default={right,left}]
\definenarrower[ConfigC][middle=2cm,default=middle]
```

Ceci est un petit texte sans effet de narrower.

```
\startConfigA
```

Ceci est un petit texte pour visualiser l'effet de narrower en ConfigA.
Ceci est un petit texte pour visualiser l'effet de narrower en ConfigA.
Ceci est un petit texte pour visualiser l'effet de narrower en ConfigA.

```
\stopConfigA
```

Ceci est un petit texte sans effet de narrower.

```
\startConfigB
```

Ceci est un petit texte pour visualiser l'effet de narrower en ConfigB.
Ceci est un petit texte pour visualiser l'effet de narrower en ConfigB.
Ceci est un petit texte pour visualiser l'effet de narrower en ConfigB.

```
\stopConfigB
```

Ceci est un petit texte sans effet de narrower.

```
\startConfigC
```

Ceci est un petit texte pour visualiser l'effet de narrower en ConfigB.
Ceci est un petit texte pour visualiser l'effet de narrower en ConfigB.
Ceci est un petit texte pour visualiser l'effet de narrower en ConfigB.

```
\stopConfigC
```

Ceci est un petit texte sans effet de narrower.

Ceci est un petit texte pour visualiser l'effet de narrower en ConfigA. Ceci
est un petit texte pour visualiser l'effet de narrower en ConfigA. Ceci est
un petit texte pour visualiser l'effet de narrower en ConfigA.

Ceci est un petit texte sans effet de narrower.

Ceci est un petit texte pour visualiser l'effet de narrower en ConfigB. Ceci
est un petit texte pour visualiser l'effet de narrower en ConfigB. Ceci est
un petit texte pour visualiser l'effet de narrower en ConfigB.

Ceci est un petit texte sans effet de narrower.

Ceci est un petit texte pour visualiser l'effet de narrower en ConfigB. Ceci
est un petit texte pour visualiser l'effet de narrower en ConfigB. Ceci est
un petit texte pour visualiser l'effet de narrower en ConfigB.

11.2 Espace vertical entre les paragraphes

11.2.1 \setupwhitespace

Comme nous le savons déjà ([section 4.2.2](#)), le nombre de lignes vides consécutives dans le fichier source n'a aucune importance pour ConTeXt : une ou plusieurs lignes vides inséreront une seule coupure de paragraphe dans le document final. Pour augmenter l'espace entre les paragraphes, il n'est d'aucune utilité d'ajouter une ligne vierge supplémentaire dans le fichier source. En revanche, cette fonction est contrôlée par la commande `\setupwhitespace` qui autorise les valeurs suivantes :

- `none` : signifie qu'il n'y aura pas d'espace vertical supplémentaire entre les paragraphes.
- `small`, `medium`, `big` : ces valeurs insèrent, respectivement, un espace vertical petit, moyen ou grand. La taille réelle de l'espace inséré par ces valeurs dépend de la taille de la police.
- `line`, `halfline`, `quarterline` : indique l'espace en terme de hauteur des lignes et insère respectivement une ligne, une demi-ligne ou un quart de ligne d'espace supplémentaire.
- `DIMENSION` : établit une dimension réelle pour l'espace entre les paragraphes. Par exemple, `\setupwhitespace[5pt]`.

En règle générale, il n'est pas conseillé de définir une dimension exacte comme valeur pour `\setupwhitespace`. Il est préférable d'utiliser les valeurs symboliques `small`, `medium`, `big`, `line`, `halfline` ou `quarterline`. Il en est ainsi pour deux raisons :

- Les valeurs symboliques `small`, `medium`, `big`, etc., sont calculées sur la base de la taille de la police, donc si celle-ci change dans certaines parties, cela changera également la quantité d'espacement vertical entre les paragraphes, et le résultat final sera toujours harmonieux. À l'inverse, une valeur fixe pour l'espacement vertical ne sera pas affectée par les changements de taille de police, ce qui se traduira normalement par un document avec des espaces blancs mal répartis (du point de vue esthétique) et non conformes aux règles de l'ajustement typographique.
- Les valeurs symboliques sont des dimensions *élastiques* (voir [section 3.8.2](#)), ce qui signifie qu'elles ont des dimensions *normales* mais qu'une certaine diminution ou augmentation de cette valeur est autorisée, afin d'aider ConTeXt dans la composition des pages de sorte que les coupures de paragraphe soient esthétiquement similaires. Mais une mesure fixe de la séparation entre les paragraphes rend plus difficile l'obtention d'une bonne pagination pour le document.

```
\setupwhitespace[medium]%
Phrase 1.1.\par
Phrase 1.2.\par
Phrase 1.3.\par
\nowhitespace
Phrase 1.4.\par
Phrase 1.5.

\setupwhitespace[big]
Phrase 2.1.\par
Phrase 2.2.\par
Phrase 2.3.\par
\nowhitespace
Phrase 2.4.\par
Phrase 2.5.
```

Phrase 1.1.
Phrase 1.2.
Phrase 1.3.
Phrase 1.4.
Phrase 1.5.
Phrase 2.1.
Phrase 2.2.
Phrase 2.3.
Phrase 2.4.
Phrase 2.5.

Une fois la valeur définie pour l'espacement vertical des paragraphes, deux commandes supplémentaires sont disponibles : `\nowhitespace`, qui élimine tout espace supplémentaire entre des paragraphes particuliers, et `\whitespace` qui fait le contraire. Cependant, ces commandes sont rarement nécessaires, car le fait est que ConTeXt gère assez bien l'espacement vertical entre les paragraphes par lui-même ; surtout si l'une des dimensions prédefinies a été insérée comme valeur, calculée à partir de la taille de police et de la hauteur de ligne courantes.

La signification de `\nowhitespace` est évidente. Mais pas nécessairement celle de `\whitespace`, car quel est l'intérêt d'ordonner l'espacement vertical pour des paragraphes particuliers étant donné que l'espacement vertical a déjà été généralement établi pour tous les paragraphes ? Cependant, lors de l'écriture de macros avancées, `\whitespace` peut être utile dans le contexte d'une boucle qui doit prendre une décision en fonction de la valeur d'une certaine condition. Il s'agit d'une programmation plus ou moins avancée, et je ne m'y attarderai pas ici.



11.2.2 Paragraphes sans espace vertical supplémentaire entre eux

Si nous voulons que certaines parties de notre document aient des paragraphes qui ne soient pas séparés par un espace vertical supplémentaire, nous pouvons bien sûr modifier la configuration générale de `\setupwhitespace`, mais cela est, d'une certaine manière, contraire à la philosophie de ConTeXt selon laquelle les commandes de configuration générale doivent être placées exclusivement dans le préambule du fichier source, afin d'obtenir une apparence générale cohérente et facilement modifiable pour les documents. D'où l'environnement « `packed` », dont la syntaxe générale est la suivante

```
\startpacked [Espace] ... \stoppacked
```

où *Espace* est un argument facultatif indiquant la quantité d'espace vertical souhaitée entre les paragraphes de l'environnement. S'il est omis, aucun espace vertical supplémentaire ne sera appliqué.

11.2.3 Ajout d'un espace vertical supplémentaire à un endroit particulier du document

Si, à un endroit particulier du document, l'espacement vertical normal entre les paragraphes n'est pas suffisant, nous pouvons utiliser la commande `\blank`. Utilisée sans argument, `\blank` insère la même quantité d'espacement vertical que celle définie avec `\setupwhitespace`. Mais nous pouvons indiquer soit une dimension spécifique entre crochets, soit l'une des valeurs symboliques calculées à partir de la taille de la police : small, medium ou big. Nous pouvons également multiplier ces tailles par un nombre entier, et ainsi de suite, par exemple, `\blank[3*medium]` insérera l'équivalent de trois sauts de ligne moyens. Nous pouvons également combiner deux tailles. Par exemple, `\blank[2*big, medium]` insérera deux sauts de ligne de grande taille et un de taille moyenne.

Puisque `\blank` est conçu pour augmenter l'espace vertical entre les paragraphes, il n'a aucun effet si un saut de page est inséré entre les deux paragraphes dont l'espacement doit être augmenté ; et si nous insérons deux ou plusieurs commandes `\blank` à la suite, seule l'une d'entre elles s'appliquera (celle dont l'espacement est le plus important). Une commande `\blank` placée après un saut de page n'a pas non plus d'effet. Toutefois, dans ces cas, nous pouvons forcer l'insertion d'un espacement vertical en utilisant le mot symbolique « `force` » comme option de la commande. Ainsi, par exemple, si nous voulons que les titres des chapitres de notre document apparaissent plus bas sur la page, de sorte que la longueur totale de la page soit inférieure à celle du reste des pages (une pratique typographique relativement fréquente), nous devons écrire dans la configuration de la commande `\chapter`, par exemple :

```
\startsection[title=C'est le  
titre]  
C'est le texte de la section .  
\stopsection
```

1 C'est le titre

C'est le texte de la section .

```
\setuphead  
  [section]  
  [page=yes,  
   before={\blank[4cm, force]},  
   after={\blank[3*medium]}]  
\startsection[title=C'est le  
titre]  
C'est le texte de la section.  
\stopsection
```

1 C'est le titre

C'est le texte de la section.

Cette séquence de commandes garantit que les sections commencent toujours sur une nouvelle page et que l'étiquette du chapitre soit positionnée quatre centimètres vers le bas. Sans l'utilisation de l'option « `force` », cela ne fonctionnera pas.

11.2.4 \setupblank et \defineblank

Plus tôt, j'ai dit que `\blank`, utilisé sans arguments, est équivalent à `\blank[big]`. Cependant, nous pouvons changer cela avec `\setupblank`, en le définissant comme `\setupblank[0.5cm]` par exemple, ou `\setupblank[medium]`. Utilisé sans argument, `\setupblank` ajustera la valeur à la taille de la police actuelle.

De même qu'avec `\setupwhitespace`, l'espace blanc inséré par `\blank`, lorsque sa valeur est l'une des valeurs symboliques prédéfinies, est une dimension élastique qui permet un certain ajustement. On peut la modifier avec « `fixed` », avec la possibilité, par la suite, de rétablir la valeur par défaut avec (« `flexible` »). Ainsi, par exemple, pour un texte en double colonne, il est recommandé de définir `\setupblank[fixed, line]`, et lors du retour à une seule colonne de revenir à la définition initiale `\setupblank[flexible, default]`.

Avec `\defineblank`, nous pouvons associer une certaine configuration à un nom. Le format général de cette commande est le suivant :

```
\defineblank [Name] [Configuration]
```

Une fois que notre configuration d'espace blanc est définie, nous pouvons l'utiliser avec `\blank[Name]`.

11.2.5 Autres procédures pour obtenir plus d'espace vertical

Dans T_EX la commande qui insère un espace vertical supplémentaire est `\vskip`. Cette commande, comme presque toutes les commandes T_EX, fonctionne également dans ConT_EXt, mais son utilisation est fortement déconseillée car elle interfère avec le fonctionnement interne de certaines macros de ConT_EXt. À sa place, il est suggéré d'utiliser `\godown` dont la syntaxe est :

```
\godown [Dimension]
```

où *Dimension* doit être un nombre avec ou sans décimales, suivi d'une unité de mesure. Par exemple, `\godown[5cm]` déplacera le prochain texte de 5 centimètres vers le bas ; toutefois, si le changement de page arrive avec l'atteinte de cette quantité, `\godown` ne fera que passer à la page suivante. De même, `\godown` n'aura aucun effet au début d'une page, bien que nous puissions *le tromper* en écrivant, par exemple « `\u\godown[3cm]` »²¹ qui va d'abord insérer un espace vide signifiant que nous ne sommes plus au début de la page, puis descendre de trois centimètres.

Comme nous le savons, `\blank` permet également d'utiliser une dimension précise comme argument. Par conséquent, du point de vue de l'utilisateur, écrire `\blank[3cm]` ou `\godown[3cm]` est pratiquement la même chose. Cependant, il existe quelques différences subtiles entre elles. Ainsi, par exemple, deux commandes `\blank` consécutives ne peuvent pas être cumulées et lorsque cela se produit, seule celle qui impose une plus grande distance est appliquée. En revanche, deux commandes `\godown` ou plus peuvent parfaitement se cumuler.

²¹ Rappellez-vous que nous utilisons le caractère « `\u` » dans ce document pour représenter un espace vide lorsqu'il est important pour nous de le voir.

```
Texte 1.  
\blank[1cm]  
Texte 2.  
\blank[1cm]  
\blank[1cm]  
Texte 3.  
\godown[1cm]  
Texte 4.  
\godown[1cm]  
\godown[1cm]  
Texte 5.
```

Texte 1.

Texte 2.

Texte 3.

Texte 4.

Texte 5.

Une autre commande TeX plutôt utile, dont l'utilisation ne pose aucun problème dans ConTeXt, est `\vfill`. Cette commande insère un espace blanc vertical flexible allant jusqu'au bas de la page (c'est le pendant vertical de `\hfill` vu [section 10.3.3](#)). C'est comme si la commande *poussait* vers le bas ce qui est écrit après elle. Cela permet d'obtenir des effets intéressants, comme placer un certain paragraphe en bas de la page, en le faisant simplement précéder de `\vfill`. Maintenant, l'effet du `\vfill` est difficile à apprécier si son utilisation n'est pas combinée avec des sauts de page forcés, car il n'y a pas beaucoup d'intérêt à pousser un paragraphe ou une ligne de texte vers le bas si le paragraphe, au fur et à mesure qu'il se développe, se développe vers le haut.

Ainsi, par exemple, pour s'assurer qu'une ligne est placée en bas de la page, il faut écrire :

```
\vfill  
Ligne en bas.  
\page[yes]
```

Comme toutes les autres commandes qui insèrent un espace vertical, `\vfill` n'a aucun effet en début de page. Mais on peut encore *le tromper* en la faisant précéder d'un espace blanc forcé, ou bien d'un `\strut`. Ainsi, par exemple :

```
\page[yes]  
\vfill  
Ligne centrée verticalement.  
\vfill
```

```
\strut\vfill  
Ligne centrée verticalement.  
\vfill
```

permet de centrer verticalement la phrase sur la page.

Un **strut**, ou *point d'appui* est un petit bloc invisible sans largeur mais avec la hauteur et la profondeur maximales d'un caractère ou d'une ligne. Nous y reviendrons plus tard, car cet élément est souvent utile dans de nombreux cas pour les auteurs et compositeur pointilleux (voir section ??).

11.3 Comment ConTeXt construit les lignes qui forment les paragraphes

L'une des principales tâches d'un système de composition consiste à prendre une longue chaîne de mots et à la diviser en lignes individuelles de la taille appropriée. Par exemple, chaque paragraphe de ce texte a été divisé en lignes de 381.26694pt de large, mais l'auteur n'a pas eu à se soucier de ces détails, car ConTeXt choisit les points d'arrêt après avoir considéré chaque paragraphe dans son intégralité, de sorte que les derniers mots d'un paragraphe peuvent réellement influencer la division de la première ligne. Par conséquent, l'espace entre les mots de l'ensemble du paragraphe est aussi uniforme que possible.

C'est l'un des aspects où l'on peut le mieux constater la différence de fonctionnement des traitements de texte et la meilleure qualité obtenue avec des systèmes tels que ConTeXt. En effet, un traitement de texte, lorsqu'il atteint la fin de la ligne et passe à la suivante, ajuste l'espace blanc de la ligne qui vient de se terminer pour permettre la justification à droite. Il fait cela pour chaque ligne, et à la fin, chaque ligne du paragraphe aura un espacement différent entre les mots. Cela peut produire un très mauvais effet (par exemple, une rivière d'espaces blancs qui traverse le texte). ConTeXt, en revanche, traite le paragraphe dans son intégralité et calcule pour chaque ligne le nombre de points d'arrêt admissibles et la quantité d'espacement entre les mots qui résulterait d'un saut de ligne. Comme le point de rupture d'une ligne affecte les points de rupture potentiels des lignes suivantes, le nombre total de possibilités peut être très élevé ; mais ce n'est pas un problème pour ConTeXt. Il prendra une décision finale en se basant sur l'ensemble du paragraphe, en s'assurant que l'espace entre les mots de chaque ligne est *aussi similaire que possible*, ce qui donne des paragraphes bien mieux composés, visuellement plus compacts.

Pour ce faire, ConTeXt teste différentes alternatives, et attribue une valeur de *mauvais goût* (*badness*) à chacune d'entre elles en fonction de ses paramètres. Ceux-ci ont été établis après une étude approfondie de l'art de la typographie. Enfin, après avoir exploré toutes les possibilités, ConTeXt choisit l'option la moins inadaptée (celle qui a la plus petite valeur de mauvais goût). En général, cela fonctionne assez bien, mais il y aura inévitablement des cas où l'on choisira des points de rupture de ligne qui ne sont pas les meilleurs, ou qui ne nous paraissent pas être les meilleurs. Par conséquent, nous voudrons parfois indiquer au programme que certains endroits ne sont pas de bons points d'arrêt. En d'autres occasions, nous voudrons forcer une rupture à un point particulier.

11.3.1 Utilisation du caractère réservé « ~ »

Les principaux candidats pour les sauts de ligne sont évidemment les espaces blancs entre les mots. Pour indiquer qu'un certain espace ne doit jamais être remplacé par un saut de ligne, nous utilisons, comme nous le savons déjà, le caractère réservé « ~ », que TeX appelle une *attache* (*tie*), liant deux mots ensemble.

Il est généralement recommandé d'utiliser cet espace insécable dans les cas suivants :

- Entre les parties qui composent une abréviation. Par exemple, U~S.
- entre les abréviations et le terme auquel elles se réfèrent. Par exemple, Dr~Anne Ruben ou p.~45.

- Entre les chiffres et le terme qui les accompagne. Par exemple, `Elizabeth~II`, `45~volumes`.
- Entre les chiffres et les symboles qui les précèdent ou les suivent, à condition qu'ils ne soient pas en exposant. Par exemple, `73~km`, `$~53` ; cependant, `35'`.
- En pourcentages exprimés en mots. Par exemple, `trente~pour~cent`.
- Dans les groupes de chiffres séparés par un espace blanc. Par exemple, `5~357~891`. Bien que dans ces cas, il soit préférable d'utiliser ce que l'on appelle l'espacement fin, obtenu dans ConTeXt avec la commande `\,`, et donc d'écrire `5\,357\,891`.
- Pour éviter qu'une abréviation soit le seul élément de cette ligne. Par exemple :

Il existe des secteurs tels que le divertissement, les médias de communication, le commerce, etc.

A ces cas, Knuth (le père de TeX) ajoute les suivantes recommandations :

- Après une abréviation qui ne se trouve pas à la fin d'une phrase.
- En référence aux parties d'un document telles que les chapitres, les annexes, les figures, etc. Par exemple : `Chapitre~12`.
- Entre le prénom et l'initiale du second nom d'une personne, ou entre l'initiale du prénom et le nom de famille. Par exemple, `Donald~E. Knuth`, `A.~Einstein`.
- Entre les symboles mathématiques en apposition aux noms. Par exemple, `dimension~d, width~w`.
- Entre des symboles en série. Par exemple : `{1, ~2, \dots, ~n$}`.
- Quand un nombre est strictement lié à une préposition. Par exemple : `de 0 à~1`.
- Lorsque des symboles mathématiques sont exprimés par des mots. Par exemple, `égale~à~n`.
- Dans les listes à l'intérieur d'un paragraphe. Par exemple : `(1)~vert`, `(2)~rouge`, `(3)~bleu`.

De nombreux cas ? Sans aucun doute, la perfection typographique a un coût en termes d'efforts supplémentaires. Il est clair que si nous ne le voulons pas, nous ne sommes pas obligés d'appliquer ces règles, mais cela ne fait pas de mal de les connaître. En outre, et je parle ici d'expérience, une fois que nous nous sommes habitués à les appliquer (ou à n'importe laquelle d'entre elles), cela devient automatique. C'est comme mettre des accents sur les mots lorsque nous les écrivons (comme nous devons le faire en français) : pour ceux d'entre nous qui le font, si nous sommes habitués à les écrire automatiquement, il ne nous faut pas plus de temps pour écrire un mot avec un accent que pour un mot sans accent.

11.3.2 La césure des mots

Sauf pour les langues composées principalement de monosyllabes, il est assez difficile d'obtenir un résultat optimal si les sauts de ligne se trouvent uniquement au niveau des espaces entre les mots. C'est pourquoi ConTeXt analyse également la possibilité d'insérer un saut de ligne entre deux syllabes d'un mot ; et pour ce faire, il est essentiel qu'il connaisse la langue du texte, puisque les règles de césure sont différentes pour chaque langue. D'où l'importance de la commande `\mainlanguage` dans le préambule du document.

Il peut arriver que ConTEXt soit incapable de couper un mot de manière appropriée. Cela peut être dû à ses propres règles de division des mots (par exemple, ConTEXt ne divise jamais un mot en deux parties si ces parties n'ont pas un nombre minimum de lettres) ou à l'ambiguïté du mot. Par exemple, le mot malaise se coupe en « malaise » s'il s'agit du nom masculin qui désigne un état de désagrément ou de gêne passagère, mais en « ma-laise » s'il s'agit du nom féminin ou de l'adjectif relatif à la Malaisie.

Quelle que soit la raison, si nous ne sommes pas satisfaits de la façon dont un mot a été scindé, ou s'il est incorrect, nous pouvons le modifier en indiquant expressément les points potentiels où un mot peut être scindé avec le symbole de contrôle `\-`. Ainsi, par exemple, si « malaise » nous posait problème, nous pourrions l'écrire dans le fichier source sous la forme « `mal\ -aise` » ou « `ma\ -laise` ».

Si le mot problématique est utilisé plusieurs fois dans notre document, il est préférable d'indiquer la manière dont il doit être séparé par un trait d'union dans notre préambule à l'aide de la commande `\hyphenation` : cette commande, qui est destinée à être incluse dans le préambule du fichier source, prend un ou plusieurs mots (séparés par des virgules) comme argument, indiquant les points auxquels ils peuvent être séparés par un trait d'union. Par exemple :

```
\hyphenation{mal-aise, roudou-dou}
```

Si le mot qui fait l'objet de cette commande ne contient pas de trait d'union, l'effet sera que le mot ne sera jamais ciselé. Ce même effet peut être obtenu en utilisant la commande `\hbox` qui crée une boîte horizontale indivisible autour du mot, ou la commande `\unhyphenated` qui empêche la césure du ou des mots qu'elle prend comme arguments. Mais alors que `\hyphenation` agit globalement, `\hbox` et `\unhyphenated` agissent localement, ce qui signifie que la commande `\hyphenation` affecte toutes les occurrences dans le document des mots inclus dans son argument, contrairement à `\hbox` ou `\unhyphenated` qui n'agissent qu'à l'endroit du fichier source où ils sont rencontrés.

En interne, le fonctionnement de la césure est contrôlé par les paramètres `\pretolerance` et `\tolerance`. La première de ces variables contrôle l'admissibilité d'une division effectuée uniquement sur un espace blanc. Par défaut, elle est égale à 100, mais si nous la modifions, par exemple, pour la porter à 10 000, alors ConTEXt considérera toujours qu'il est acceptable qu'il y ait un saut de ligne qui n'implique pas la séparation des mots en fonction des syllabes, ce qui signifie que *de facto*, nous supprimons la césure basée sur les syllabes. Alors que si, par exemple, nous fixions la valeur de `\pretolerance` à -1, nous obligerions ConTEXt à utiliser la césure des mots en fin de ligne à chaque fois.

Nous pouvons directement définir une valeur arbitraire pour `\pretolerance` en lui attribuant simplement une valeur dans notre document. Par exemple :

```
\pretolerance=10000
```

Nous pouvons également manipuler cette valeur avec les valeurs « `lesshyphenation` » et « `morehyphenation` » dans `\setupalign`. Voir à ce sujet la section [section 11.6.1](#).

11.3.3 Niveau de tolérance pour les sauts de ligne

Lorsqu'il recherche les points de retour à la ligne possibles, ConTeXt est généralement assez strict, ce qui signifie qu'il préfère permettre à un mot de dépasser la marge de droite parce qu'il n'a pas pu y placer de césure, et préfère ne pas insérer de retour à la ligne avant le mot si cela entraîne une augmentation trop importante de l'espace inter-mots sur cette ligne. Ce comportement par défaut donne normalement des résultats optimaux, et ce n'est qu'exceptionnellement que certaines lignes ressortent quelque peu sur la droite. L'idée est que l'auteur (ou le compositeur) examine ces cas exceptionnels une fois le document terminé, pour prendre la décision appropriée, qui pourrait être une commande `\break` devant le mot qui s'étend au-delà, ou pourrait également signifier une formulation différente du paragraphe afin que ce mot change de position.

Cependant, dans certains cas, la faible tolérance de ConTeXt peut être un problème. Dans ces cas, nous pouvons lui demander d'être plus tolérant avec les espaces blancs dans les lignes. Nous disposons pour cela de la commande `\setuptolerance`, qui nous permet de modifier le niveau de tolérance dans le calcul des sauts de ligne, que ConTeXt appelle « tolérance horizontale » (car elle affecte l'espace horizontal) et « tolérance verticale » lors du calcul des sauts de page. Nous en parlerons dans la section 11.6.2.

La tolérance horizontale (qui est celle qui affecte les sauts de ligne), est fixée à la valeur « `verystrict` » par défaut. Nous pouvons modifier cela en définissant, comme alternatives, l'une des valeurs suivantes : « `strict` », « `tolerant` », « `verytolerant` » ou « `stretch` ». Ainsi, par exemple, « `verytolerant` » rendra presque impossible le dépassement de la marge de droite par une ligne, quitte à établir un espacement très important et inesthétique entre les mots d'une ligne.

Exemple avec « `verytolerant` »

Pour en revenir à l'utilisation des caractères dans l'édition électronique, bon nombre de nouveaux typographes

tirent leurs connaissances et leurs informations sur les règles de la typographie des livres, des magazines in-

formatiques ou des manuels d'instruction qu'ils reçoivent à l'achat d'un PC ou d'un logiciel.

Exemple avec « `verystrict` »

Pour en revenir à l'utilisation des caractères dans l'édition électronique, bon nombre de nouveaux typographes

tirent leurs connaissances et leurs informations sur les règles de la typographie des livres, des magazines informatiques ou des ma-

nuels d'instruction qu'ils reçoivent à l'achat d'un PC ou d'un logiciel.

11.3.4 Forcer un saut de ligne à un certain point

Pour forcer un retour à la ligne à un certain point, nous utilisons les fonctions `\break`, `\crlf` ou `\\"`. La première d'entre elles, `\break`, introduit un saut de ligne à l'endroit où elle se trouve. Cela entraînera très probablement une déformation esthétique de la ligne où la commande est placée, avec une immense quantité d'espace blanc entre les mots de cette ligne. Comme on peut le voir dans l'exemple suivant, où la commande `\break` dans la troisième ligne (du fragment source à gauche) donne lieu à une deuxième ligne assez laide (dans le texte formaté à droite).

Au coin du vieux quartier, je l'ai vu `\emph{se pavaner}` comme le font`\break` les gros bras quand ils marchent, les mains toujours dans les poches de leur pardessus, pour que personne ne puisse savoir lequel d'entre eux porte le poignard.

Au coin du vieux quartier, je l'ai vu *se pavaner* comme le font les gros bras quand ils marchent, les mains toujours dans les poches de leur pardessus, pour que personne ne puisse savoir lequel d'entre eux porte le poignard.

Pour éviter cet effet, nous pouvons utiliser les commandes `\\"` ou `\crlf` qui insèrent également un saut de ligne forcé, mais elles remplissent la ligne d'origine avec suffisamment d'espace vide pour l'aligner à gauche :

Au coin du vieux quartier, je l'ai vu `\emph{se pavaner}` comme le font`\\"` les gros bras quand ils marchent, les mains toujours dans les poches de leur pardessus, pour que personne ne puisse savoir lequel d'entre eux porte le poignard.

Au coin du vieux quartier, je l'ai vu *se pavaner* comme le font les gros bras quand ils marchent, les mains toujours dans les poches de leur pardessus, pour que personne ne puisse savoir lequel d'entre eux porte le poignard.

Sur les lignes *normales*, pour autant que je sache, il n'y a pas de différence entre `\\"` et `\crlf` ; mais dans un titre de section, il y a une différence :

- `\\"` génère un saut de ligne dans le corps du document, mais pas lorsque le titre de la section est transféré dans la table des matières.
- `\crlf` génère un retour à la ligne qui s'applique à la fois dans le corps du document et lorsque le titre de la section est transféré dans la table des matières.

Un saut de ligne ne doit pas être confondu avec un saut de paragraphe. Un saut de ligne met simplement fin à la ligne en cours et commence la ligne suivante, mais nous maintient dans le même paragraphe, de sorte que la séparation entre la ligne d'origine et la nouvelle ligne sera déterminée par l'espacement normal dans un paragraphe. Par conséquent, il n'y a que trois scénarios dans lesquels il peut être recommandé de forcer un saut de ligne :

- Dans des cas très exceptionnels, lorsque ConTeXt n'a pas été en mesure de trouver un saut de ligne approprié, de sorte qu'une ligne dépasse sur la droite. Dans ces cas (qui se produisent très rarement, principalement lorsque la ligne contient des *boîtes indivisibles*, ou du texte *verbatim* [voir section 10.2.4]), il peut être utile de forcer un saut de ligne avec `\break` juste avant le mot qui dépasse dans la marge de droite.
- Dans les paragraphes qui sont en fait composés de lignes individuelles, chacune contenant des informations indépendantes de celles des lignes précédentes, par exemple, l'en-tête d'une lettre dans laquelle la première ligne peut contenir le nom de l'expéditeur, la deuxième le destinataire et la troisième la date ; ou dans un texte parlant de la paternité d'une œuvre, où une ligne contient le nom de l'auteur, une autre sa fonction ou sa position académique et peut-être une troisième ligne avec la date, etc. Dans ces cas, le saut de ligne doit être forcé avec les commandes `\&` ou `\crlf`. Il est également courant que ce type de paragraphe soit aligné à droite.
- Lors de la rédaction de poèmes ou d'autres types de textes similaires, pour séparer un vers d'un autre. Bien que dans ce dernier cas, il soit préférable d'utiliser l'environnement `lines` expliqué dans section 11.5.1.

11.4 Espace interligne

L'interligne est la distance séparant les lignes qui composent un paragraphe. ConTeXt calcule automatiquement cette distance en fonction de la police utilisée et, surtout, de la taille de base définie avec `\setupbodyfont` ou `\switchtobodyfont`.

Plus précisément, l'espace interligne est déterminée à partir de la taille de la police utilisée. Elle vaut dans `2.8ex`, un `ex` étant la hauteur du caractère x. Une ligne a une hauteur et une profondeur. La somme de la hauteur maximale et de la profondeur maximale donne la distance interligne.

```
\leavevmode
\ruledhbox{%
\tfc Une ligne a une {\tt height} et une {\tt depth} ~:
\blackrule[height=max,depth=0pt] +
\blackrule[height=0pt,depth=max] =
\blackrule[height=max,depth=max]}%
```

Une ligne a une `height` et une `depth` : 

Nous pouvons influencer l'espace interligne avec la fonction `\setupinterlinespace` qui permet trois types de syntaxe différents :

- `\setupinterlinespace [espace interligne]`, où *espace interligne* est une valeur précise ou un mot symbolique qui attribue un espace interligne prédéfini :
 - Lorsqu'il s'agit d'une valeur précise, il peut s'agir d'une dimension (par exemple, `15pt`), ou d'un nombre simple, entier ou décimal (par exemple, `1,2`). Dans ce dernier cas, le nombre est interprété comme « nombre de lignes » en fonction de l'interligne par défaut de ConTeXt.
 - Lorsqu'il s'agit d'un mot symbolique, il peut s'agir de « `small` », « `medium` » ou « `big` », chacun appliquant respectivement un espace interligne petit, moyen ou grand, toujours basé sur l'espace interligne par défaut que ConTeXt appliquerait.
- `\setupinterlinespace [.,.,.=.,.]`. Dans ce mode, l'espacement interligne est défini en modifiant explicitement les mesures sur la base desquelles ConTeXt calcule l'espacement interligne approprié. Dans ce mode, l'espacement est défini en modifiant explicitement les mesures sur la base desquelles ConTeXt calcule l'espacement approprié. J'ai dit précédemment que l'interligne est calculé sur la base de la police spécifique et de sa taille, mais c'était pour simplifier les choses : en fait, la police et la taille servent à établir certaines mesures sur la base desquelles l'interligne est calculé. Grâce à cette approche `\setupinterlinespace`, ces mesures sont modifiées et, par conséquent, l'espace interligne l'est aussi. Les mesures et valeurs réelles qui peuvent être manipulées par cette procédure (dont je n'expliquerai pas la signification car cela dépasse le cadre d'une simple introduction) sont les suivantes : `line`, `height`, `depth`, `minheight`, `mindepth`, `distance`, `top`, `bottom`, `stretch` et `shrink`.

Les valeurs des paramètres `top` et `bottom` déterminent la hauteur de la première ligne et la profondeur de la dernière ligne sur la page. Ce sont également des ratios mais cette fois appliqués à la taille de la police de corps de texte (`bodyfont`). Ces paramètres sont en rapport avec les paramètres T_EX `\topskip` et `\maxdepth`.

- `\setupinterlinespace [Nom]`. Avec ce mode, nous établissons ou configurons un type d'interligne spécifique et personnalisé, préalablement défini avec `\defineinterlinespace`.

Les paramètres par défauts sont les suivants :

```
\setupinterlinespace  
[height=.72,  
 depth=.28,  
 top=1.0,  
 bottom=0.4,  
 line=2.8ex]
```

Voyons un exemple

```
\startbuffer[textureinterligne]  
Bordeaux est une commune du Sud-Ouest de la France. Elle est capitale  
de la Gaule aquitaine dès le début du IIIe siècle. La ville est  
connue dans le monde entier pour les vins et vignobles du Bordelais.  
\stopbuffer
```

```
\defineparagraphs[TroisColumns][n=3,distance=0.04\textwidth]  
  
\startTroisColumns  
%  
\getbuffer[textureinterligne]  
\TroisColumns  
%  
\setupinterlinespace[1.25]  
\getbuffer[textureinterligne]  
\TroisColumns  
%  
\setupinterlinespace[1.5]  
\getbuffer[textureinterligne]  
%\stopTroisColumns
```

Bordeaux est une commune du Sud-Ouest de la France. Elle est capitale de la Gaule aquitaine dès le début du III^e siècle. La ville est connue dans le monde entier pour les vins et vignobles du Bordelais.

Bordeaux est une commune du Sud-Ouest de la France. Elle est capitale de la Gaule aquitaine dès le début du III^e siècle. La ville est connue dans le monde entier pour les vins et vignobles du Bordelais.

Bordeaux est une commune du Sud-Ouest de la France. Elle est capitale de la Gaule aquitaine dès le début du III^e siècle. La ville est connue dans le monde entier pour les vins et vignobles du Bordelais.

Avec `\defineinterlinespace` nous pouvons associer une certaine configuration d'espace interligne à un nom spécifique que nous pouvons ensuite simplement déclencher à un moment donné dans notre document avec `\setupinterlinespace[Nom]`. Pour revenir à un espace interligne normal, nous devrions alors écrire `\setupinterlinespace[reset]`.

```
\defineinterlinespace [Nom] [Configuration]
```

11.5 Autres questions relatives aux lignes

11.5.1 Conversion des sauts de ligne du fichier source en sauts de ligne du document final

Comme nous le savons déjà (voir [section 4.2.2](#)), par défaut, ConTeXt ignore les sauts de ligne du fichier source qu'il considère comme de simples espaces vides, sauf s'il y a deux ou plusieurs sauts de ligne consécutifs, auquel cas un saut de paragraphe sera inséré. Cependant, dans certaines situations, il peut être intéressant de respecter les sauts de ligne du fichier source original tels qu'ils ont été placés, par exemple, lors de l'écriture de poèmes. Pour cela, ConTeXt nous offre l'environnement « `lines` » dont le format est :

```
\startlines [Options] ... \stoplines
```

où les options peuvent être l'une des suivantes, entre autres :

- **space** : Lorsque cette option est définie avec la valeur « `on` », en plus de respecter les sauts de ligne du fichier source, l'environnement respectera également les espaces vides du fichier source, en ignorant temporairement la règle d'absorption.
- **before** : Texte ou commande à exécuter avant d'entrer dans l'environnement.
- **after** : Texte ou commande à exécuter après avoir quitté l'environnement.
- **inbetween** : Texte ou commande à exécuter lors de l'entrée dans l'environnement.
- **indenting** : Valeur indiquant si les paragraphes doivent être indentés ou non dans l'environnement (voir [section 11.1.1](#)).
- **align** : Alignement des lignes dans l'environnement (voir [section 11.6](#)).
- **style** : Commande de style à appliquer dans l'environnement.
- **color** : Couleur à appliquer dans l'environnement.

Ainsi par exemple :

```
\startlines
One-one was a race horse.
Two-two was one too.
One-one won one race.
Two-two won one too.
\stoplines
```

```
One-one was a race horse.
Two-two was one too.
One-one won one race.
Two-two won one too.
```

Nous pouvons également modifier le fonctionnement par défaut de l'environnement avec `\setuplines` et, comme pour de nombreuses commandes de ConTeXt, il est également possible d'attribuer un nom à une configuration particulière de cet environnement. Nous le faisons avec la commande `\definelines` dont la syntaxe est :

```
\definelines [Nom] [Configuration]
```

où, en tant que configuration, nous pouvons inclure les mêmes options que celles qui ont été expliquées de manière générale pour l'environnement. Une fois que nous avons défini notre environnement de ligne personnalisé, nous devons écrire pour l'insérer :

```
\startlines[Nom] ... \stoplines
```

11.5.2 Numérotation des lignes

Dans certains types de textes, il est courant d'établir une certaine forme de numérotation des lignes, par exemple dans les textes sur la programmation informatique où il est relativement courant que les fragments de code proposés à titre d'exemple aient leurs lignes numérotées, ou encore dans les poèmes, les éditions critiques, etc. Pour toutes ces situations, ConTeXt offre l'environnement `linumbering` dont le format est le suivant

```
\startlinenumbering[Options] ... \stoplinenumbering
```

Les options disponibles sont les suivantes :

- **continue** : Dans le cas où il y a plusieurs parties de notre document nécessitant une numérotation des lignes, cette option fait en sorte que la numérotation recommence pour chaque partie (« `continue=no` », la valeur par défaut). En revanche, si la numérotation des lignes doit se poursuivre là où la partie précédente s'est arrêtée, nous choisissons « `continue=yes` ».
- **start** : Indique le numéro de la première ligne dans les cas où nous ne voulons pas qu'il soit « 1 », ou qu'il corresponde à l'énumération précédente.
- **step** : Toutes les lignes incluses dans l'environnement seront numérotées, mais, au moyen de cette option, nous pouvons indiquer que le numéro n'est imprimé qu'à certains intervalles. Dans le cas des poèmes, par exemple, il est courant que le numéro n'apparaisse que par multiples de 5 (vers 5, 10, 15...).

Toutes ces options peuvent être indiquées, en général pour tous les environnements `linenumbering` de notre document, avec `\setuplinenumbering`. Cette commande nous permet également de configurer d'autres aspects de la numérotation des lignes :

- **conversion** : Type de numérotation des lignes. Il peut s'agir de l'un de ceux expliqués sur page 172 concernant la numérotation des chapitres et des sections.
- **style** : Commande (ou commandes) déterminant le style qu'aura la numérotation des lignes (police, taille, variante...).
- **color** : Couleur dans laquelle le numéro de ligne sera imprimé.
- **location** : localisation du numéro de ligne (lieu où il sera placé). Il peut s'agir de l'un des éléments suivants : `text`, `begin`, `end`, `default`, `left`, `right`, `inner`, `outer`, `inleft`, `inright`, `margin`, `inmargin`.
- **distance** : Distance entre le numéro de la ligne et la ligne elle-même.
- **align** : Alignement du numéro. Peut être : `inner`, `outer`, `flushleft`, `flushright`, `left`, `right`, `middle` ou `auto`.

- **command** : Commande à laquelle le numéro de ligne sera transmis en tant que paramètre avant l'impression.
- **width** : Largeur réservée à l'impression du numéro de ligne.
- **left, right, margin** :

Nous pouvons également créer différentes configurations personnalisées de numérotation des lignes avec `\definelinenumbering` de sorte que la configuration soit associée à un nom :

```
\definelinenumbering [Name] [Configuration]
```

Une fois qu'une configuration spécifique a été définie et associée à un nom, nous pouvons l'utiliser avec la commande

```
\startlinenumbering [Name] ... \stoplinenumbering
```

11.6 Alignement horizontal et vertical

²² Par largeur *exacte*, j'entends la largeur de la ligne *avant*. ConTeXt ajuste la taille de l'espace inter-mots pour permettre la justification.

La commande qui contrôle l'alignement du texte en général est `\setupalign`. Cette commande est utilisée pour contrôler l'alignement horizontal et vertical.

11.6.1 Alignement horizontal

Lorsque la largeur *exacte* d'une ligne de texte n'occupe pas toute la largeur possible, cela pose le problème de savoir que faire avec l'espace blanc qui en résulte.²² Nous pouvons essentiellement faire trois choses à cet égard :

1. L'accumuler sur l'un des deux côtés de la ligne : si nous l'accumulons sur le côté gauche, la ligne semblera *poussée* vers la droite, tandis que si nous l'accumulons sur le côté droit, la ligne reste sur le côté gauche. On parle, dans le premier cas, d'un *alignement à droite* et, dans le second, d'un *alignement à gauche*. Par défaut, ConTeXt applique l'alignement à gauche à la dernière ligne des paragraphes. Lorsque plusieurs lignes consécutives sont alignées à gauche, le côté droit est irrégulier ; mais lorsque l'alignement est à droite, le côté qui semble irrégulier est le gauche. Pour nommer les options qui alignent l'un ou l'autre côté, ConTeXt ne définit pas le côté où elles sont alignées, mais le côté où elles sont irrégulières. Par conséquent, l'option `flushright` entraîne un alignement à gauche et l'option `flushleft` un alignement à droite. En tant qu'abréviations de `flushright` et `flushleft`, `\setupalign` prend également en charge les valeurs `right` et `left`. Mais **attention** : ici, le sens des mots est trompeur. Même si `left` signifie « `left` » et `right` signifie « `right` », `\setupalign[left]` s'aligne à droite et `\setupalign[right]` s'aligne à gauche. Au cas où le lecteur se demanderait pourquoi ce commentaire a été fait, il serait utile de citer le wiki ConTeXt : « ConTeXt utilise les options `flushleft` et `flushright`. Les alignements à droite et à gauche sont inversés par rapport aux directions habituelles dans toutes les commandes qui acceptent une option d'alignement, dans le sens de « `ragged left` » et « `ragged right` ». Malheureusement, lorsque Hans a écrit cette partie de ConTeXt pour la première fois, il pensait à l'alignement « `ragged right` » et « `ragged left` », plutôt qu'à « `flush left` » et « `flush right` ». Et maintenant qu'il en est ainsi depuis un certain temps, il est impossible de le modifier, car cela romprait la rétrocompatibilité avec tous les documents existants qui l'utilisent. » Dans les documents préparés pour une impression recto-verso, outre les marges droite et gauche, il existe également des marges intérieure et extérieure. Les valeurs `flushinner` (ou simplement `inner`) et `flushouter` (ou simplement `outer`) établissent l'alignement correspondant dans ces cas.
2. Le répartir sur les deux marges. Le résultat sera que la ligne est centrée. L'option `\setupalign` qui permet de le faire est `middle`.
3. La répartir entre tous les mots qui composent la ligne, si nécessaire en augmentant l'espace entre les mots, afin que la ligne ait exactement la même largeur que l'espace dont elle dispose. Dans ces cas, on parle de *lignes justifiées*. C'est également la valeur par défaut de ConTeXt, c'est pourquoi il n'y a pas d'option spéciale dans `\setupalign` pour l'établir. Cependant, si nous avons modifié l'alignement justifié par défaut, nous pouvons le rétablir avec `\setupalign[reset]`.

```

\startalignment [flushleft]
Une phrase pour voir l'impact de
l'option d'alignement {\bf \tt
flushleft}.

\stopalignment
\startalignment [right]
Une phrase pour voir l'impact de
l'option d'alignement {\bf \tt
right}.

\stopalignment
\startalignment [flushright]
Une phrase pour voir l'impact de
l'option d'alignement {\bf \tt
flushright}.

\stopalignment
\startalignment [left]
Une phrase pour voir l'impact de
l'option d'alignement {\bf \tt
left}.

\stopalignment

```

Une phrase pour voir l'impact de l'option d'alignement `flushleft`.

Une phrase pour voir l'impact de l'option d'alignement `right`.

Une phrase pour voir l'impact de l'option d'alignement `flushright`.

Une phrase pour voir l'impact de l'option d'alignement `left`.

La valeur de `\setupalign` que nous venons de voir (`right`, `flushright`, `left`, `flushleft`, `inner`, `flushinner`, `outer`, `flushouter` et `middle`) peut être combinée avec `broad`, ce qui donne un alignement un peu plus grossier.

Deux autres valeurs possibles de `\setupalign` qui affectent l'alignement horizontal, concernent la césure des mots en fin de ligne, car le fait que cela soit fait ou non dépend de la mesure exacte de la ligne, qui est plus ou moins grande, ce qui affecte l'espace blanc restant.

À cet effet, `\setupalign` autorise la valeur `morehyphenation` qui fait travailler ConTeXt plus fort pour trouver des points d'arrêt basés sur la césure, et `lesshyphenation` qui produit l'effet inverse. Avec `\setupalign[horizontal, morehyphenation]`, l'espace blanc restant dans les lignes sera réduit et l'alignement sera donc moins apparent. Au contraire, avec `\setupalign[horizontal, lesshyphenation]`, il restera plus d'espace blanc, et l'alignement sera plus visible.

`\setupalign` est destiné à être inclus dans le préambule et à affecter tout le document ou, à être inclus à un point spécifique et à affecter tout à partir de ce point jusqu'à la fin. Si nous voulons seulement changer l'alignement d'une ou plusieurs lignes, nous pouvons utiliser :

- L'environnement « `alignment` », destiné à affecter plusieurs lignes. Son format général est le suivant, où *Options* sont toutes celles autorisées pour `\setupalign` :

```
\startalignment [Options] ... \stopalignment
```

- `\leftaligned`, `\midaligned` ou `\rightaligned` provoquent respectivement un alignement à gauche, centré ou à droite ; et si nous voulons que le dernier mot d'un paragraphe (mais seulement celui-ci et pas le reste de la ligne) soit aligné à droite, nous pouvons utiliser `\wordright` (voir l'exemple en fin de section 10.3.3). Toutes ces commandes requièrent que le texte à affecter se trouve entre des accolades.

Notez, d'autre part, que si les mots « `right` » et « `left` » dans `\setupalign` provoquent l'alignement opposé à ce que le nom suggère, il n'en va pas de même avec les commandes `\leftaligned` et `\rightaligned` qui provoquent exactement le type d'alignement que leur nom suggère : `left` à gauche, et `right` à droite.

11.6.2 Alignement vertical

Si l'alignement horizontal intervient lorsque la largeur d'une ligne n'occupe pas tout l'espace disponible, l'alignement vertical concerne la hauteur de toute la page : si la hauteur *exacte* du texte d'une page n'occupe pas toute la hauteur disponible, que fait-on de l'espace blanc restant ? Toujours avec `\setupalign` Nous pouvons l'empiler en haut (« `height` »), ce qui signifie que le texte de la page sera poussé vers le bas ; nous pouvons l'empiler en bas (« `bottom` ») ou le répartir entre les paragraphes (« `line` »). La valeur par défaut de l'alignement vertical est « `bottom` ».

L'option « `height` » semble pas fonctionner.

Niveau de tolérance en vertical

De la même manière que nous pouvons modifier le niveau de tolérance de ConTeXt en ce qui concerne la quantité d'espace horizontal autorisée dans une ligne (tolérance horizontale) avec `\setuptolerance`, nous pouvons également modifier sa tolérance verticale, c'est-à-dire la tolérance pour un espace inter-paragraphe plus grand que celui que ConTeXt considère comme raisonnable par défaut pour une page correctement composée. Les valeurs possibles pour la tolérance verticale sont les mêmes que pour la tolérance horizontale : `verystrict`, `strict`, `tolerant` et `verytolerant`. La valeur par défaut est `\setuptolerance [vertical, strict]`.

Contrôler les veuves et les orphelins

Un aspect qui affecte indirectement l'alignement vertical est le contrôle des veuves et des orphelins. Ces deux termes font référence aux lignes de paragraphes unique et isolées du reste du paragraphe à cause d'un saut de page. Ceci n'est pas considéré comme typographiquement approprié. Si la ligne séparée du reste du paragraphe est la première de la page, on parle d'une *ligne veuve* ; si la ligne séparée de son paragraphe est la dernière de la page, on parle d'une *ligne orpheline*.

Par défaut, ConTeXt n'implémente pas de contrôle pour s'assurer que ces lignes ne se produisent pas. Mais nous pouvons changer cela en modifiant certaines des variables internes de ConTeXt : `\widowpenalties` contrôle les lignes veuves et `\clubpenalties` contrôle les lignes orphelines. Ainsi, les déclarations suivantes dans le préambule de notre document garantiront que ce contrôle est effectué :

```

\startsetups [mypenalties]
\setdefaultpenalties
\setpenalties\widowpenalties{3}{10000} %      3 == nombre de lignes
\setpenalties\clubpenalties {3}{10000} % 10000 == penalité entre les 3 lignes
\stopsetups

\setuplayout [setups=mypenalties]

```

Par l'exécution de ces commandes ConTeXt évitera d'insérer un saut de page qui sépare les 3 premières ou dernières lignes d'un paragraphe de la page sur laquelle se trouve le reste. Cet évitement sera plus ou moins rigoureux selon la valeur que nous attribuons aux variables. Avec une valeur de 10 000, comme celle que j'ai utilisée dans l'exemple, le contrôle sera absolu ; avec une valeur de 150, par exemple, le contrôle ne sera pas aussi rigoureux et il se peut qu'il y ait parfois quelques lignes veuves ou orphelines lorsque l'alternative est pire en termes typographiques.

Chapitre 12

Constructions et paragraphes spéciaux

Table of Contents : [12.1 Notes de bas de page et de fin de document](#) ; [12.1.1 Le types de notes dans ConTeXt et les commandes associées](#) ; [12.1.2 Zoom les notes de bas de page et de fin de document](#) ; [12.1.3 Notes locales](#) ; [12.1.4 Crédit et utilisation de types de notes personnalisées](#) ; [12.1.5 Configurer les notes](#) ; [12.1.6 Exclusion temporaire des notes lors de la compilation](#) ; [12.2 Paragraphes avec plusieurs colonnes](#) ; [12.2.1 L'environnement \startcolumns](#) ; [12.2.2 Paragraphes parallèles](#) ; [12.3 Listes structurées](#) ; [12.3.1 Sélection du type de liste et du séparateur entre les items de la liste](#) ; A Listes non ordonnées ; B Listes ordonnées ; [12.3.2 Saisie des éléments d'une liste](#) ; [12.3.3 Configuration basique des listes](#) ; [12.3.4 Configuration complémentaire des listes](#) ; [12.3.5 Listes simples à l'aide de la commande \items.](#) ; [12.3.6 Prédétermination du comportement des listes et création de nos propres types de listes](#) ; [12.4 Descriptions et énumérations](#) ; [12.4.1 Descriptions](#) ; [12.4.2 Énumérations](#) ; [12.5 Lignes et cadres](#) ; [12.5.1 Lignes simples](#) ; [12.5.2 Lignes liées au texte](#) ; [12.5.3 Mots ou textes encadrés](#) ; A Premiers exemples ; B Retour à la ligne et alignement ; C Largeur ; D Positionnement par rapport à la ligne de base ; E Attention au paramètre strut ; [12.6 Autres environnements et constructions d'intérêt](#) ;

12.1 Notes de bas de page et de fin de document

Les notes sont des « éléments textuels secondaires employés à des fins diverses, comme clarifier ou prolonger le texte principal, fournir la référence bibliographique des sources, y compris les citations, renvoyer à d'autres documents ou énoncer le sens du texte ». [Libro de Estilo de la Lengua española (Guide de style de la langue espagnole), p. 195]. Elles sont particulièrement importantes dans les textes de nature académique. Elles peuvent être placées à différents endroits de la page ou du document. Aujourd'hui, les plus répandues sont celles qui sont situées en bas de page (appelées, par conséquent, notes de bas de page) ; parfois, elles sont également situées dans l'une des marges (notes de marge), à la fin de chaque chapitre ou section, ou à la fin du document (notes de fin de document). Dans les documents particulièrement complexes, il peut également y avoir différentes séries de notes : notes de

l'auteur, notes du traducteur, mises à jour, etc. En particulier, dans les éditions critiques, l'appareil de notes peut devenir assez complexe et seuls quelques systèmes de composition sont capables de le supporter. ConTeXt est l'un d'entre eux. De nombreuses commandes sont disponibles pour établir et configurer les différents types de notes.

Pour expliquer cela, il est utile de commencer par indiquer les différents éléments qui peuvent être impliqués dans une note :

- *Marque* ou *note ancre* : Signe placé dans le corps du texte pour indiquer qu'une note lui est associée. Tous les types de notes ne sont pas associés à une *ancre*, mais lorsqu'il y en a une, cette *ancre* apparaît à deux endroits : à l'endroit du texte principal auquel la note fait référence, et au début du texte de la note elle-même. C'est la présence de la même marque de référence à ces deux endroits qui permet d'associer la note au texte principal.
- La note *ID* ou *identifiant* : La lettre, le chiffre ou le symbole qui identifie la note et la distingue des autres notes. Certaines notes, par exemple les notes de marge, peuvent ne pas avoir d'*ID*. Lorsque ce n'est pas le cas, l'*identifiant* coïncide normalement avec l'*ancre* de la note.
Si nous pensons exclusivement aux notes de bas de page, nous ne verrons aucune différence entre ce que je viens d'appeler une *Marque* et l'*id*. Nous voyons clairement la différence dans d'autres types de notes : Les notes de ligne, par exemple, ont un *id*, mais pas de marque de référence.
- *Texte* ou *Contenu* de la note, toujours situé à un endroit différent sur la page ou dans le document que la commande qui génère la note et indique son contenu.
- *Étiquette* associée à la note : Étiquette ou nom associé à une note qui n'apparaît pas dans le document final, mais qui permet d'y faire référence et de retrouver son identifiant ailleurs dans le document.

12.1.1 Le types de notes dans ConTeXt et les commandes associées

Nous disposons de différents types de notes dans ConTeXt. Pour l'instant, je me contenterai de les énumérer, en les décrivant en termes généraux et en fournissant des informations sur les commandes qui les génèrent. Plus tard, je développerai les deux premiers :

- **Notes de bas de page (footnotes)** : Sans aucun doute la plus populaire, à tel point qu'il est courant que tous les types de notes soient désignés par le terme générique de *notes de bas de page*. Les notes de bas de page introduisent une *marque* avec l'*id* de la note à l'endroit du document où se trouve la commande, et insèreront le texte de la note elle-même en bas de la page où la marque apparaît. Elles sont créées avec la commande `\footnote`.
- **Notes de fin de document (endnotes)** : Ces notes, créées avec la commande `\endnote`, sont insérées à l'endroit du document où se trouve une marque portant l'*identifiant* de la note ; mais le contenu de la note est inséré à un autre endroit du document, et l'insertion est produite par une commande différente (`\placenotes`).

- **Notes marginales (margin notes)** : Comme leur nom l'indique, elles sont écrites dans la marge du texte et il n'y a pas d'identifiant ou de marque ou d'ancre générée automatiquement dans le corps du document. Les deux principales commandes (mais pas les seules) qui les créent sont `\inmargin` et `\margintext` (voir section 5.7).
- **Notes de ligne (line notes)** : Type de note typique des environnements où les lignes sont numérotées, comme dans le cas de `\startlinenumbering ... \stoplinenumbering` (voir section ??). La note, qui est généralement écrite en bas, fait référence à un numéro de ligne spécifique. Elles sont générées par la commande `\linenote` qui est configurée avec `\setuplinenote`. Cette commande n'imprime pas de *marque* dans le corps du texte, mais dans la note elle-même, elle imprime le numéro de ligne auquel la note se réfère (utilisé comme *ID*).

Je vais maintenant développer exclusivement les deux premiers types de notes :

- Les notes de marge sont traitées ailleurs (section 5.7).
- Les notes de ligne ont un usage très spécialisé (notamment dans les éditions critiques) et je pense que dans un document d'introduction comme celui-ci, il suffit que le lecteur sache qu'elles existent.

Cependant, pour le lecteur intéressé, je recommande une vidéo (en espagnol) accompagnée d'un texte (également en espagnol) sur les éditions critiques dans ConTeXt, dont l'auteur est Pablo Rodríguez. Elle est disponible [en ligne](#). Il est également très utile pour comprendre plusieurs des paramètres généraux des notes en général.

12.1.2 Zoom les notes de bas de page et de fin de document

La syntaxe des commandes `\footnotes` et `\endnotes` et les mécanismes de configuration et de personnalisation dont elles disposent sont assez similaires, puisque, en réalité, ces deux types de notes sont des instances particulières d'une construction plus générale (notes), dont d'autres instances peuvent être définies avec la commande `\definenote` (voir section 12.1.4).

La syntaxe de la commande qui crée chacun de ces types de notes est la suivante :

```
\footnote [Étiquette] {Texte}
\endnote [Étiquette] {Texte}
```

- *Étiquette* est un argument facultatif qui attribue à la note une étiquette qui nous permettra d'y faire référence ailleurs dans le document.
- *Texte* est le contenu de la note. Il peut être aussi long que nous le souhaitons, et inclure des paragraphes et des paramètres spéciaux, bien qu'il faille noter qu'en ce qui concerne les notes de bas de page, une mise en page correcte est assez difficile dans les documents contenant des notes abondantes et excessivement longues.

En principe, toute commande qui peut être utilisée dans le texte principal peut être utilisée dans le texte de la note. Cependant, j'ai pu vérifier que certaines constructions et caractères qui ne posent aucun problème dans le texte principal, génèrent une erreur de compilation lorsqu'ils sont utilisés dans le texte de la note. J'ai trouvé ces cas lors de mes tests, mais je ne les ai pas organisés de quelque manière que ce soit.

Lorsque l'argument *Étiquette* a été utilisé pour définir une étiquette pour la note, la commande `\note` nous permet de récupérer l'ID de la note en question. Cette commande affiche l'ID de la note associée à l'étiquette qu'elle prend en argument sur le document. Ainsi, par exemple :

```
\setuppapersize[A7,landscape]
\setupbodyfont[8pt]
\starttext
Humpty Dumpty[\footnote[humpty]{probablement le personnage de comptine
anglais le plus célèbre}] s'est assis sur un mur, Humpty Dumpty\note[humpty]
a fait une grande chute. Tous les chevaux du roi et tous les hommes du
roi n'ont pas pu remettre Humpty\note[humpty] ensemble.
\stoptext
```

1

Humpty Dumpty¹ s'est assis sur un mur, Humpty Dumpty¹ a fait une grande chute. Tous les chevaux du roi et tous les hommes du roi n'ont pas pu remettre Humpty¹ ensemble.

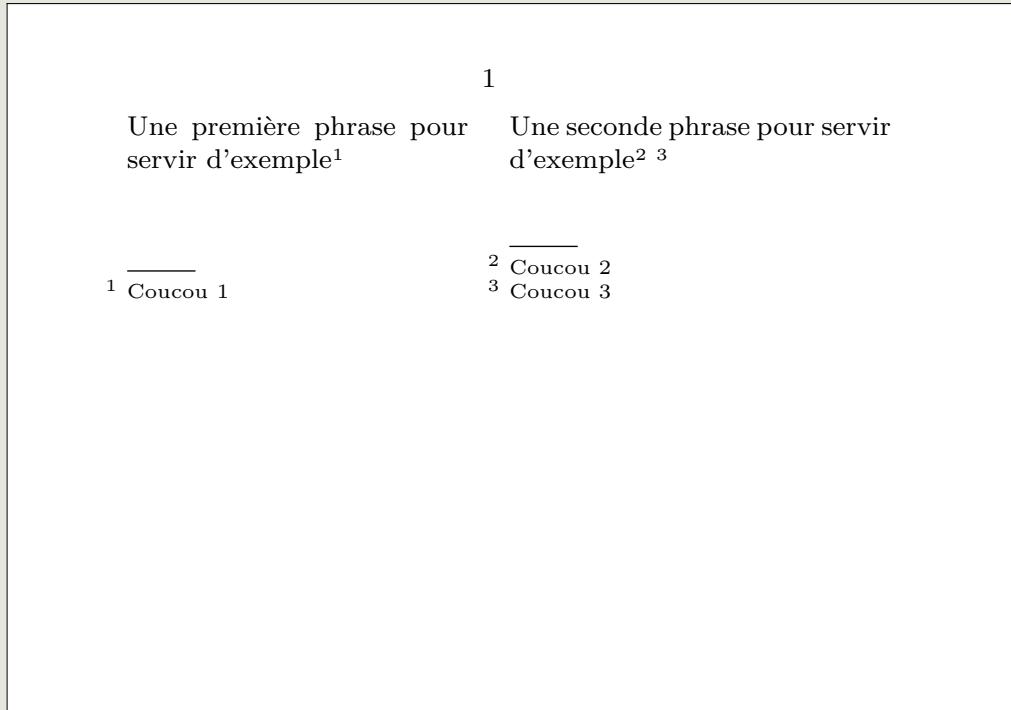
¹ probablement le personnage de comptine anglais le plus célèbre



La principale différence entre `\footnote` et `\endnote` est l'endroit où la note apparaît :

\footnote En règle générale, il imprime le texte de la note au bas de la page sur laquelle se trouve la commande, de sorte que la marque de la note et son texte (ou le début du texte, s'il doit être réparti sur deux pages) apparaissent sur la même page. Pour ce faire, ConTEXt fera les ajustements nécessaires lors de la composition de la page en calculant l'espace requis par l'emplacement de la note au bas de la page.

`\footnotetext` fait la même chose que `\footnote` mais sans introduire la marque localement. La marque peut-être affichée séparément avec `\note`, il faut pour cela passer aux deux commandes le nom de référence entre crochet « [MonÉtiquette] ». Une utilisation de `\footnotetext` en combinaison avec `\note` serait des notes à l'intérieur d'autres notes. C'est ainsi qu'a été construit la référence 3 dans l'exemple suivant, où vous pouvez constater que dans un environnement `\startcolumns` les notes de bas de page deviennent des notes de bas de colonne :



```
Une seconde phrase pour servir d'exemple
\footnote{Coucou 2}
\footnotetext[testici]{Coucou 3}
\note[testici]
```

`\endnote` imprime uniquement l'ancre de la note à l'endroit du fichier source où elle se trouve. Le contenu réel de la note est inséré à un autre endroit du document à l'aide d'une autre commande, (`\placenotes [endnote]`) qui, à l'endroit où elle se trouve, insère le contenu de *toutes* les notes de fin de document (ou du chapitre ou de la section en question).

12.1.3 Notes locales

L'environnement `\startlocalfootnotes` signifie que les notes de bas de page qui y sont incluses sont considérées comme des notes *locales*, ce qui signifie que leur numérotation sera réinitialisée et que le contenu des notes ne sera pas automatiquement inséré avec le reste des notes, mais seulement à l'endroit du document où se trouvera la commande `\placelocalfootnotes`, qui peut ou non se trouver dans l'environnement.

Ceci¹ est une phrase avec des notes imbriquées.

Petit A	Petit B ⁴
---------	----------------------

Table 1 Table hors localfootnotes

Petit C	Petit D ¹
¹ coucou D	

Table 2 Table encadrée par localfootnotes

¹ ou cela², si vous préférez.

² ou encore possiblement celle-ci³.

³ pourrait être totalement différentes.

⁴ coucou B



```
\placetable[Table encadrée par localfootnotes]
{\startlocalfootnotes
\bTABLE[width=0.25\textwidth]
\bTR \bTD Petit C \eTD \bTD Petit D\footnote{coucou D} \eTD \eTR
\bTR \bTD[nc=2,frame=off] \placelocalfootnotes \eTD \eTR
\eTABLE
\stoplocalfootnotes}
```

12.1.4 Crédit et utilisation de types de notes personnalisées

Nous pouvons créer des types spéciaux de notes avec la commande `\definenote`. Cela peut être utile dans les documents complexes où il y a des notes de différents auteurs, ou à des fins différentes, pour distinguer graphiquement chacun des types de notes dans notre document au moyen d'un format différent et d'une numérotation différente.

La syntaxe de la commande `\definenote` est la suivante :

```
\definenote[MesNotes][Modèle][Configuration]
```

- *MesNotes* est le nom que nous attribuons à notre nouveau type de note.
- *Modèle* est le modèle de note qui sera utilisé initialement, soit par exemple `\footnote` et `\endnote`. Dans le premier cas, notre modèle de note fonctionnera comme des notes de bas de page, et dans le second cas, comme des notes de fin de document. Nous les définirons dans le fichier source avec la commande `\MesNotes` et nous utiliseront `\placenotes [MesNotes]` (le nom que nous avons attribué à ces types de notes) à l'endroit où nous souhaitons les afficher dans le document final.
- *Configuration* est un argument facultatif qui nous permet de distinguer notre nouveau type de notes de son modèle : soit en définissant un format différent, soit un type de numérotation différent, soit les deux.

Selon la liste officielle des commandes ConTeXt (voir [section 3.6](#)), les paramètres qui peuvent être fournis lors de la création du nouveau type de note sont basés sur ceux qui pourraient être fournis plus tard avec `\setupnote`. Cependant, comme nous le verrons bientôt, il existe en fait deux commandes possibles pour configurer les notes : `\setupnote` et `\setupnotation` (voir [section ??](#)). Je pense donc qu'il est préférable d'omettre cet argument lors de la création du type de note, puis de configurer nos nouvelles notes à l'aide des commandes appropriées. Au moins, c'est plus facile à expliquer.

Par exemple, pour créer un nouveau type de note appelé « MesNotesBleues » qui sera similaire aux notes de bas de page mais dont le contenu sera imprimé en gras et en bleu, nous pouvons définir ainsi :

```
\definenote [MesNotes]
\setupnotation [MesNotesBleues] [color=blue, style=bf]
```

Une fois créée, nous disposons de la commande `\MesNotesBleues` dont la syntaxe est similaire à `\footnote`.

Coucou¹

¹ [coucou ici](#)



12.1.5 Configurer les notes

La configuration des notes (notes de bas de page ou de fin de page, notes créées avec `\definenote` et aussi notes de ligne mises en place avec `\linenote`) est réalisée avec deux commandes : `\setupnote` et `\setupnotation`.

`\setupnote` possède 35 options de configuration *directes* et 45 options supplémentaires héritées de `\setupframed` ; `\setupnotation` possède 45 options de configuration *directes* et 23 autres héritées de `\setupcounter` `\setupframed`. Comme ces options ne sont pas documentées et, bien que pour beaucoup d'entre elles nous puissions deviner leur utilité à partir de leur nom, nous devons vérifier si notre intuition est vraie ou non ; et aussi en tenant compte du fait que beaucoup de ces options permettent un certain nombre de valeurs et qu'elles doivent toutes être testées... Vous verrez que pour écrire cette explication j'ai dû faire un certain nombre de tests ; et bien que faire un test soit rapide, faire beaucoup de tests est lent et ennuyeux. J'espère donc que le lecteur m'excusera si je lui dis qu'en dehors des deux commandes de configuration générale des notes que je mentionne dans le texte principal et sur lesquelles je me concentre dans l'explication suivante, je laisserai de côté quatre autres possibilités de configuration dans l'explication :

- `\setupnotes` et `\setupnotations` : En d'autres termes, le même nom mais au pluriel. Le wiki dit que les versions singulière et plurielle de la commande sont synonymes, et je le crois.
- `\setupfootnotes` et `\setupendnotes` : Nous supposons qu'il s'agit d'applications spécifiques pour, respectivement, les notes de bas de page et les notes de fin. Il serait peut-être plus facile d'expliquer la configuration des notes sur la base de ces commandes, cependant, puisque je n'ai pas réussi à faire fonctionner la première option (`numberconversion`) que j'ai essayée avec `\setupfootnotes`, bien que je sache que les autres options de ces commandes fonctionnent... J'étais trop paresseux pour ajouter les tests nécessaires pour inclure ces deux commandes dans l'explication aux nombreux tests que j'ai déjà dû faire pour écrire ce qui suit.

Mais je suis d'avis (d'après les quelques tests aléatoires que j'ai effectués) que tout ce qui fonctionne dans ces deux commandes, mais dont je laisse l'explication de côté, fonctionne également dans les commandes pour lesquelles je donne une explication.

La syntaxe est similaire dans les deux cas :

```
\setupnote [NoteType] [Configuration]
\setupnotation [NoteType] [Configuration]
```

où *NoteType* fait référence au type de note que nous configurons (`footnote`, `end-note` ou le nom d'un type de note que nous avons nous-mêmes créé), et *configuration* contient les options de configuration particulières de la commande.

Le problème est que les noms de ces deux commandes n'indiquent pas clairement la différence entre elles ni ce que chacune d'elles configure ; et le fait que de nombreuses options de ces commandes ne soient pas documentées n'aide pas beaucoup non plus. Après de nombreux tests, je n'ai pas été en mesure de parvenir à une conclusion qui me permettrait de comprendre pourquoi certaines choses sont configurées avec l'une, tandis que d'autres le sont avec l'autre,²³ sauf peut-être que, en raison des choix que j'ai faits pour le faire fonctionner, `\setupnotation` affecte toujours le texte de la note, ou l'ID qui est imprimé avec le texte de la note, alors que `\setupnote` a quelques options qui affectent la marque de la note insérée dans le texte principal.

Je vais maintenant essayer d'organiser ce que j'ai découvert après avoir fait quelques tests avec les différentes options des deux commandes. Je laisse de côté la plupart des options des deux commandes, car elles ne sont pas documentées et je n'ai pas pu tirer de conclusions quant à leur utilité ou aux conditions dans lesquelles elles doivent être utilisées :

- **ID utilisé pour la note :** Les notes sont toujours identifiées par un numéro. Ce que nous pouvons configurer ici est :
 - *Le premier nombre* : contrôlé par `start` dans `\setupnotation`. Sa valeur doit être un nombre entier, que ConTeXt utilise comme point de départ pour comptabiliser les notes.
 - *Le système de numérotation*, qui dépend de l'option `numberconversion` dans `\setupnotation`. Ses valeurs peuvent être :
 - * *chiffres arabes* : `n`, `N` ou `numbers`.
 - * *chiffres romains* : `I`, `R`, `Romannumerals`, `i`, `r`, `romannumerals`. Les trois premiers sont des chiffres romains en majuscules et les trois derniers en minuscules.
 - * *Numérotation avec des lettres* : `A`, `Character`, `Characters`, `a`, `character`, `characters` selon que l'on souhaite que les lettres soient en majuscules (les trois premières options) ou en minuscules (le reste).

²³ Il existe une page dans le [ConTeXt wiki](#) que j'ai découverte par hasard (puisque elle n'est pas spécifiquement dédiée aux notes), qui suggère que la différence est que `\setupnote` contrôle le texte de la note à insérer, et `\setupnotation` l'environnement de la note dans laquelle elle sera placée (?) Mais ceci n'est pas cohérent avec le fait que, par exemple, la largeur du texte de la note (qui a à voir avec son *insertion*) est contrôlée par l'option `width` de `\setupnote` et non par l'option `\setupnotation` du même nom. Ce qui est contrôlé dans ce cas c'est la largeur de l'espace pris pour afficher la marque, avant l'affichage du texte de la note.

- * *Numérotation avec des mots*. En d'autres termes, on écrit le mot qui désigne le nombre et ainsi, par exemple, « 3 » devient « trois ». Deux méthodes sont possibles. L'option `Words` écrit les mots en majuscules et `words` en minuscules.
 - * *Numérotation avec symboles* : on peut utiliser quatre jeux de symboles différents selon l'option choisie : `set 0`, `set 1`, `set 2` ou `set 3`. Sur page ??, vous trouverez un exemple des symboles utilisés dans chacune de ces options.
 - *L'événement qui détermine la remise à zéro de la numérotation des notes* : Cela dépend de l'option `way` dans `\setupnotation`. Lorsque la valeur est `bytext`, toutes les notes du document seront numérotées séquentiellement sans que la numérotation soit réinitialisée. Lorsqu'elle vaut `bychapter`, `bysection`, `bysubsection`, etc., le compteur de notes sera réinitialisé chaque fois qu'un nouveau chapitre, ou nouvelle section ou sous-section est créé, tandis que lorsqu'elle vaut `byblock`, la numérotation sera réinitialisée chaque fois que nous changerons de bloc dans la macrostructure du document (voir [section 7.6](#)). La valeur `bypage` fait redémarrer le compteur de notes à chaque fois que la page est changée.
- **Configurer la marque de la note :**
- Affichage ou non : Option `number` dans `\setupnotation`.
 - Positionnement de la marque par rapport au texte de la note : L'option `alternative` dans `\setupnotation` : elle peut prendre l'une des valeurs suivantes : `left`, `inleft`, `leftmargin`, `right`, `inright`, `rightmargin`, `inmargin`, `margin`, `innermargin`, `outermargin`, `serried`, `hanging`, `top`, `command`. Par défaut la marque est dans l'espace entre la marge et le texte. Avec `hanging` la marque est intégré dans la zone de texte, de même avec `serried` qui en plus intègre une identification lors des retours à la ligne du texte de la note .
 - Style et couleur à appliquer à la marque dans le corps du texte : L'option `textstyle` et `textcolor` dans `\setupnote`.
 - Style et couleur à appliquer à la marque dans la note elle-même : L'option `headstyle` et `headcolor` dans `\setupnotation`.
 - Commande à appliquer à la marque dans le corps du texte : L'option `textcommand` dans `\setupnote`.
 - Commande à appliquer à la marque dans la note elle-même : L'option `numbercommand` dans `\setupnotation`.
- Les options `numbercommand` et `textcommand` doivent consister en une commande qui prend le contenu de la marque comme argument. Il peut s'agir d'une commande auto-définie. Cependant, j'ai constaté que les commandes de formatage simples (`\bf`, `\it`, etc.) fonctionnent, bien qu'elles ne soient pas des commandes devant prendre un argument.
- Distance entre la marque et le texte (dans la note elle-même) : l'option `distance` dans `\setupnotation`. Elle est complétée par l'option `width` qui indique la largeur à accorder à l'affichage de la marque elle-même.
 - Existence ou non d'un lien hypertexte permettant de sauter entre la marque dans le texte principal et la marque dans la note elle-même : L'option `interaction` dans `\setupnote`. Avec `yes` comme valeur, il y aura un lien, et avec `no` il n'y en aura pas.

■ **Configuration du texte de la note elle-même.** Nous pouvons influencer les aspects suivants :

- Positionnement : cela dépend de l'option `location` dans `\setupnote`.

En principe, nous savons déjà que les notes de bas de page sont placées en bas de la page (`location=page`) et les notes de fin de page au point où la commande `\placenotes[endnote]` (`location=text`) est trouvée. (`location=text`), mais nous pouvons ajuster cette fonction et définir les notes de bas de page, par exemple, en tant que `location=text`. Les notes de bas de page fonctionneront alors de la même manière que les notes de fin de document et apparaîtront à l'endroit du document où se trouve la commande `\placenotes[footnote]`, ou la commande spécifique aux notes de bas de page `\placefootnotes`. Avec cette procédure, nous pourrions, par exemple, imprimer les notes sous le paragraphe dans lequel elles se trouvent.

- Séparation des paragraphes entre les notes : par défaut, chaque note est imprimée dans son propre paragraphe, mais on peut faire en sorte que toutes les notes d'une même page soient imprimées dans le même paragraphe en définissant l'option `paragraph` de `\setupnote` sur « yes ».
- Style et couleur dans lequel le texte de la note sera écrit : l'option `style` et `color` dans `\setupnotation`.
- Taille de la lettre : l'option `bodyfont` dans `\setupnote`.

Cette option ne concerne que le cas où l'on souhaite définir manuellement une taille de police pour les notes de bas de page. Ce n'est presque jamais une bonne idée de le faire car, par défaut, ConTeXt ajuste la taille de la police des notes de bas de page pour qu'elle soit plus petite que celle du texte principal, mais avec une taille proportionnelle à celle de la police du corps principal.

- Marge gauche pour le texte de la note : l'option `margin` dans `\setupnotation`.
- Largeur maximale : l'option `width` dans `\setupnote`.
- Nombre de colonnes : l'option `n` de `\setupnote` détermine si le texte de la note sera sur deux colonnes ou plus. La valeur « `n` » doit être un nombre entier.

■ **Espace entre les notes ou entre les notes et le texte :** ici, nous avons les options suivantes :

- `rule`, dans `\setupnote` détermine s'il y aura ou non une ligne (règle) entre la zone des notes et la zone de la page contenant le texte principal. Ses valeurs possibles sont `yes`, `on`, `no` et `off`. Les deux premières valeurs activent la règle et la dernière la désactive. Il est possible de configurer la ligne plus précisément en indiquant `rule=command` et en définissant `rulecommand` par exemple

```
rule=command,  
rulecommand={\blackrule [width=0.5\textwidth,color=green,height=3pt,depth=-2pt]},
```

- `before`, dans `\setupnotation` : commande ou commandes à exécuter avant l'insertion du texte de la note. Sert à insérer un espacement supplémentaire, des lignes de séparation entre les notes, etc.
- `after`, dans `\setupnotation` : commande ou commandes à exécuter après l'insertion du texte de la note.

Finissons par un exemple :

Ce début de phrase *One* est suivi d'une fin de phrase *Two*.

```
One coucou ici ceci est
une note de bas de page qui s'étend
tout au long de la page
Two coucou là ceci est une
note de bas de page qui s'étend tout
au long de la page
```



12.1.6 Exclusion temporaire des notes lors de la compilation

Les commandes `\notesenabledfalse` et `\notesenabledtrue` indiquent à ConTeXt d'activer ou de désactiver la compilation des notes respectivement. Cette fonction peut être utile si l'on souhaite obtenir une version sans notes lorsque le document comporte des notes nombreuses et étendues. Dans mon expérience personnelle, par exemple, lorsque je corrige une thèse de doctorat, je préfère la lire une première fois en une seule fois, sans les notes, puis faire une seconde lecture avec les notes incorporées.

12.2 Paragraphes avec plusieurs colonnes

La composition du texte peut être effectuée en plusieurs colonnes :

- a. Comme une caractéristique générale de la mise en page.
- b. Comme une caractéristique de certaines constructions telles que, par exemple, les listes structurées, les notes de bas de page ou les notes de fin de document.
- c. Comme une caractéristique appliquée à des paragraphes particuliers d'un document.

Dans tous ces cas, la plupart des commandes et des environnements fonctionneront parfaitement même si nous travaillons avec plus d'une colonne. Il existe cependant quelques limitations, principalement en ce qui concerne les objets flottants en général (voir [section 13.1](#)) et avec les tableaux en particulier ([section 13.3](#)) même s'ils ne sont pas flottants.

En ce qui concerne le nombre de colonnes autorisées, ConTeXt n'a pas de limite théorique. Cependant, il existe des limites physiques qui doivent être prises en compte :

- la largeur du papier : un nombre illimité de colonnes nécessite une largeur illimitée de papier (si le document est destiné à être imprimé) ou d'écran (s'il s'agit d'un document destiné à être affiché sur écran). En pratique, compte tenu de la largeur *normale* des formats de papier commercialisés et utilisés pour composer les livres, et des écrans des appareils informatiques, il est difficile qu'un texte soit composé selon plus de quatre ou cinq colonnes.
- la taille de la mémoire de l'ordinateur : le manuel de référence de ConTeXt indique que, sur des systèmes *normaux* (ni particulièrement puissants, ni particulièrement limités en ressources), il est possible de gérer entre 20 et 40 colonnes.

Dans cette section, je me concentrerai sur l'utilisation du mécanisme multi-colonnes dans les paragraphes ou fragments spéciaux, car

- Les colonnes multiples comme option de mise en page ont déjà été abordées (dans la [sous-section B de la section 5.3.4](#)).
- La possibilité offerte par certaines constructions, telles que les listes structurées ou les notes de bas de page, de composer du texte sur plus d'une colonne, est discutée en fonction de la construction ou de l'environnement en question.

12.2.1 L'environnement `\startcolumns`

La procédure normale pour insérer des parties de texte composées en plusieurs colonnes consiste à utiliser l'environnement `columns` dont le format est :

```
\startcolumns[Configuration] ... \stopcolumns
```

où *Configuration* nous permet de contrôler de nombreux aspects de l'environnement. Nous pouvons indiquer la configuration désirée à chaque fois que nous appelons l'environnement, ou adapter le fonctionnement par défaut de l'environnement pour tous les appels à l'environnement, ce dernier point pouvant être réalisé avec

`\setupcolumns[Configuration]`

Dans les deux cas, les options de configuration sont les mêmes. Les plus importantes, classées selon leur fonction, sont les suivantes :

- **Options permettant de contrôler le nombre de colonnes et l'espace entre elles :**
 - `n` : contrôle le nombre de colonnes. Si cette option est omise, deux colonnes seront générées.
 - `nleft`, `nright` : ces options sont utilisées dans la mise en page de documents recto-verso (voir [subsection A](#) de section ??), pour établir le nombre de colonnes sur les pages de gauche (paires) et de droite (impaires) respectivement.
 - `distance` : espace entre les colonnes.
 - `separator` : détermine ce qui marque la séparation entre les colonnes. Il peut s'agir de `espace` (valeur par défaut) ou `rule`, auquel cas une ligne (règle) sera générée entre les colonnes. Dans le cas où une règle est établie entre les colonnes, cette règle peut à son tour être configurée avec les deux options suivantes :
 - * `rulecolor` : couleur du trait.
 - * `rulethickness` : épaisseur du trait.
 - `maxwidth` : Largeur maximale que peuvent avoir les colonnes + l'espace entre elles.
- **Options qui contrôlent la distribution du texte entre les colonnes :**
 - `balance` : par défaut, ConTeXt équilibre (*balance*) les colonnes, c'est-à-dire qu'il répartit le texte entre elles afin qu'elles aient plus ou moins la même quantité de texte. Cependant, nous pouvons définir cette option avec l'option « no » : le texte ne commencera pas dans une colonne tant que la précédente ne sera pas pleine.
 - `direction` : détermine dans quel sens le texte est réparti entre les colonnes. Par défaut, l'ordre de lecture naturel est suivi (de gauche à droite), mais si vous donnez à cette option la valeur `reverse`, l'ordre de lecture sera de droite à gauche.
- **Options affectant la composition du texte dans l'environnement :**
 - `tolerance` : un texte écrit sur plus d'une colonne signifie que la largeur de ligne à l'intérieur d'une colonne est plus petite, et comme expliqué lors de la description du mécanisme utilisé par ConTeXt pour construire les lignes ([section 11.3](#)), cela rend difficile la localisation des points optimaux pour insérer les sauts de ligne. Cette option nous permet de modifier temporairement la tolérance horizontale d'un environnement (voir [section 11.3.3](#)), afin de faciliter la composition du texte et éviter les débordements.
 - `align` : contrôle l'alignement horizontal des lignes dans l'environnement. Il peut prendre l'une des valeurs suivantes : `right`, `flushright`, `left`, `flushleft`, `inner`, `flushinner`, `outer`, `flushouter`, `middle` ou `broad`. Pour connaître la signification de ces options, voir [section 11.6.1](#).
 - `color` : spécifie le nom de la couleur dans laquelle le texte de l'environnement sera écrit.

Parfois, pour positionner les flottants (des tableaux, des images, voir section 13) dans un contexte de composition en colonne, dans le cas où ces flottants couvrent toute la largeur des colonnes, il est préférable de commencer par quitter l'environnement `columns` avec `\stopcolumns`, de positionner le flottant avec par exemple avec `\startplacefigure`, et de reprendre ensuite avec `\stopplacefigure` puis `\startcolumns`.

12.2.2 Paragraphes parallèles

Une version spécifique de la composition en plusieurs colonnes met en parallèle les paragraphes. Dans ce type de construction, le texte est réparti sur deux colonnes (généralement, mais parfois plus de deux), mais il n'est pas autorisé à circuler librement entre elles, et garde au contraire un contrôle strict sur ce qui apparaîtra dans chaque colonne. Ceci est très utile, par exemple, pour générer des documents qui mettent en contraste deux versions d'un texte, comme la nouvelle et l'ancienne version d'une loi récemment modifiée, ou dans des éditions bilingues ; ou encore pour rédiger des glossaires pour des définitions de textes spécifiques où le texte à définir apparaît à gauche et la définition à droite, etc.

Normalement, nous utiliserions le mécanisme de table pour traiter ce type de paragraphes, mais cela est dû au fait que la plupart des processeurs de texte ne sont pas aussi puissants que ConTeXt qui possède les commandes `\defineparagraphs` et `\setupparagraphs` qui construisent ce type de paragraphe en utilisant le mécanisme de colonne, qui, bien qu'il ait des limites, est plus flexible que le mécanisme de table.

Pour autant que je sache, ces paragraphes n'ont pas de nom particulier. Je les ai appelés « Paragraphes parallèles » parce que cela me semble être un terme plus descriptif que celui que ConTeXt utilise pour s'y référer : « paragraphes ».

Les commandes de base ici sont `\defineparagraphs` et `\setupparagraphs` dont la syntaxe est :

```
\defineparagraphs [Nom] [Configuration]
\setupparagraphs [Nom] [NumColonne] [Configuration]
```

où *Nom* est le nom donné au nouvel environnement en question, *NumColonne* est un argument facultatif permettant de configurer une colonne particulière, et *Configuration* permet de déterminer son fonctionnement pratique. Par exemple :

```

\defineparagraphs [MonTrio]      [n=3, before={\blank},after={\blank}]

\setupparagraphs [MonTrio] [1] [width=.1\textwidth, style=bold]
\setupparagraphs [MonTrio] [2] [width=.4\textwidth]

\startMonTrio
825
\MonTrio
Fondation de la ville de Murcie.
\MonTrio
Les origines de la ville de Murcie sont incertaines, mais il est prouvé qu'il a été ordonné de la fonder sous le nom de Madina (ou Medina) en l'an 825 par l'émir d'al-Àndalus Abderramán II, probablement sur un établissement beaucoup plus ancien.
\stopMonTrio

```

825

Fondation de la ville de Murcie.

Les origines de la ville de Murcie sont incertaines, mais il est prouvé qu'il a été ordonné de la fonder sous le nom de Madina (ou Medina) en l'an 825 par l'émir d'al-Àndalus Abderramán II, probablement sur un établissement beaucoup plus ancien.

Le fragment ci-dessus crée un environnement à trois colonnes appelé « MonTrio », puis configure la première colonne pour qu'elle occupe 10% de la largeur de la ligne et soit écrite en gras, et configure la deuxième colonne pour qu'elle occupe 40% de la largeur de la ligne. Comme la troisième colonne n'est pas configurée, elle aura la largeur restante, c'est-à-dire 50%. Une fois l'environnement créé et configuré, nous pouvons l'utilisons.

Si nous voulions continuer à raconter l'histoire de Murcie, une nouvelle instance de l'environnement (`\startMonTrio`) serait nécessaire pour l'événement suivant, car il n'est pas possible d'inclure plusieurs *rangées* avec ce mécanisme.

De l'exemple qui vient d'être donné, je voudrais souligner les détails suivants :

- Une fois l'environnement créé avec, par exemple, `\defineparagraphs [MaryPoppins]`, celui-ci devient un environnement normal qui commence par `\startMaryPoppins` et se termine par `\stopMaryPoppins`.
- Une commande `\MaryPoppins` est également créée, utilisée dans l'environnement pour indiquer quand changer de colonne.

En ce qui concerne les options de configuration des paragraphes parallèles (`\setupparagraphs`), je comprends qu'à ce stade de l'introduction, et compte tenu de l'exemple qui vient d'être donné, le lecteur est déjà prêt à comprendre l'objectif de chacune des options, et je me contenterai donc d'indiquer ci-dessous le nom et le type des options et, le cas échéant, les valeurs possibles. N'oubliez pas, cependant,

que `\setupparagraphs [Nom] [Configuration]` établit des configurations qui affectent tout l'environnement, tandis que `\setupparagraphs [Nom] [NumColonne] [Configuration]` applique des configurations exclusivement à la colonne indiquée.

- | | | |
|---|--|---|
| <ul style="list-style-type: none">■ <code>n</code> : Nombre de colonne■ <code>before</code> : Commande à exécuter avant l'environnement■ <code>after</code> : Commande à exécuter après l'environnement■ <code>width</code> : Dimension, largeur de la colonne | <ul style="list-style-type: none">■ <code>distance</code> : Dimension, distance entre les colonnes■ <code>align</code> : Similaire à <code>\setupalign</code>■ <code>inner</code> : Commande■ <code>rule</code> : on off■ <code>rulethickness</code> : Dimension, épaisseur du trait de séparation | <ul style="list-style-type: none">■ <code>rulecolor</code> : Couleur du trait■ <code>style</code> : Style à appliquer au texte■ <code>color</code> : Couleur à appliquer au texte |
|---|--|---|

La liste des options ci-dessus n'est pas complète ; j'ai exclu de la liste des options celles que je n'expliquerai pas normalement ici. J'ai également profité du fait que nous nous trouvions dans la section consacrée aux colonnes pour afficher la liste des options en triple colonne, bien que je ne l'aie pas fait avec l'une des commandes expliquées dans cette section, mais avec l'option `columns` de l'environnement `itemize`, auquel la section suivante est consacrée.

12.3 Listes structurées

²⁴ En typographie, un retrait qui s'applique à toutes les lignes d'un paragraphe sauf la première est appelé un *tiret suspendu*, ce qui permet de trouver facilement le premier mot ou caractère du paragraphe.

Lorsque les informations sont présentées de manière ordonnée, elles sont plus faciles à saisir pour le lecteur. Mais si l'agencement est également perceptible visuellement, cela souligne pour le lecteur le fait que nous avons ici un texte structuré. C'est pourquoi il existe certaines *constructions* ou *mécanismes* qui tentent de mettre en évidence la disposition visuelle du texte, contribuant ainsi à sa structuration. Parmi les outils que ConTEXt met à la disposition de l'auteur à cette fin, le plus important, qui fait l'objet de cette section, est l'environnement `\itemize` qui permet de développer ce que nous pourrions appeler des *listes structurées*.

Les listes sont des séquences d'*éléments de texte* (que j'appellerai *items*), chacun d'entre eux étant précédé d'un caractère qui permet de le mettre en évidence en le différenciant du reste, et que j'appellerai le « séparateur ». Le séparateur peut être un chiffre, une lettre ou un symbole. Généralement (mais pas toujours), les *items* sont des paragraphes et la liste est formatée de manière à assurer la *visibilité* du séparateur pour chaque élément, généralement en appliquant un retrait suspendu qui le met en évidence.²⁴ Dans le cas de listes imbriquées, l'indentation de chacune d'elles augmente progressivement. Le langage HTML appelle généralement les listes où le séparateur est un nombre ou un caractère qui augmente de manière séquentielle, des *listes ordonnées*, ce qui signifie que chaque *item* de la liste aura un séparateur différent qui nous permettra de nous référer à chaque élément par son numéro ou son identifiant ; et il donne le nom de *listes non ordonnées* à celles où le même caractère ou symbole est utilisé pour chaque élément de la liste.

ConTEXt gère automatiquement la numérotation ou l'ordre alphabétique du séparateur dans les listes numérotées, ainsi que l'indentation que doivent avoir les listes imbriquées ; et, dans le cas de listes non ordonnées imbriquées, il s'occupe également de la sélection d'un caractère ou d'un symbole différent qui permet de distinguer d'un coup d'œil le niveau d'un *item* dans la liste en fonction du symbole qui le précède.

Le manuel de référence dit que le niveau maximum d'imbrication dans les listes est de 4, mais je suppose que c'était le cas en 2013, lorsque le manuel a été écrit. D'après mes tests, il ne semble pas y avoir de limite à l'imbrication des listes *ordonnées* (dans mes tests, j'ai atteint jusqu'à 15 niveaux d'imbrication). Pour les listes non ordonnées, il ne semble pas y avoir de limite non plus, dans le sens où peu importe le nombre d'imbrications que nous incluons, aucune erreur ne sera générée ; mais, pour les listes non ordonnées, ConTEXt applique seulement des symboles par défaut pour les neuf premiers niveaux d'imbrication.

Quoi qu'il en soit, il convient de souligner qu'un nombre excessif d'imbrications de listes peut avoir l'effet inverse de celui recherché, à savoir que le lecteur se sent perdu, incapable de situer chaque élément dans la structure générale de la liste. C'est pourquoi je pense personnellement que, bien que les listes soient un outil puissant pour structurer un texte, il n'est presque jamais bon de dépasser le troisième niveau d'imbrication ; et même le troisième niveau ne devrait être utilisé que dans certains cas où nous pouvons le justifier.

L'outil général pour écrire des listes dans ConTEXt est l'environnement `\itemize` dont la syntaxe est la suivante :

The general tool for writing lists in ConTEXt is the `\itemize` environment whose syntax is as follows :

```
\startitemize[Options][Configuration] ... \stopitemize
```

où les deux arguments sont facultatifs. Le premier permet d'utiliser des noms symboliques comme contenu auquel une signification précise a été attribuée par ConTeXt ; le second argument, rarement utilisé, permet d'attribuer des valeurs spécifiques à certaines variables qui affectent le fonctionnement de l'environnement.

12.3.1 Sélection du type de liste et du séparateur entre les *items* de la liste

A. Listes non ordonnées

Par défaut, la liste générée par `itemize` est une liste non ordonnée, dans laquelle le séparateur sera automatiquement sélectionné en fonction du niveau d'imbrication :

- Pour le premier niveau d'imbrication.
- Pour le second niveau d'imbrication.
- * Pour le troisième niveau d'imbrication.
- > Pour le quatrième niveau d'imbrication.
- Pour le cinquième niveau d'imbrication.
- Pour le sixième niveau d'imbrication.
- Pour le septième niveau d'imbrication.
- Pour le huitième niveau d'imbrication.
- ✓ Pour le neuvième niveau d'imbrication.

Cependant, nous pouvons indiquer expressément que nous voulons que le symbole associé à un niveau particulier soit utilisé, simplement en passant le numéro du niveau comme argument. Ainsi, `\startitemize[4]` générera une liste non ordonnée dans laquelle le caractère `>`, sera utilisé comme séparateur, quel que soit le niveau d'imbrication de la liste.

Nous pouvons également modifier le symbole prédéterminé pour chaque niveau avec `\definesymbol` :

```
% definesymbol
% [Level]
% [Symbol associated with the level]
\definesymbol [symA] [\diamond]
\definesymbol [symB] [(\blackrule[height=1.3ex, width=0.9ex, depth=-0.4ex,])]
\setupitemize [1] [packed] [color=middlegreen, symbol=symA]
\setupitemize [2] [packed] [color=middlered, symbol=symB]
\startitemize
\item Texte 1 \item Texte 2
\startitemize \item Texte 3 \item Texte 4 \stopitemize
\item Texte 5
\stopitemize
```

- ◆ Texte 1
- ◆ Texte 2
- Texte 3
- Texte 4
- ◆ Texte 5

B. Listes ordonnées

Pour générer une liste ordonnée, nous devons indiquer à `\itemize` le type d'ordre que nous voulons. Cela peut être :

n	1, 2, 3, 4, ...	a	a, b, c, d, ...	r	i, ii, iii, iv, ...
m	1, 2, 3, 4, ...	A	A, B, C, D, ...	R	I, II, III, IV, ...
g	$\alpha, \beta, \gamma, \delta, \dots$	KA	A, B, C, D, ...	KR	I, II, III, IV, ...
G	A, B, Γ, Δ, \dots				

La différence entre **n** et **m** réside dans la police utilisée pour représenter le nombre : **n** utilise la police activée à ce moment-là, tandis que **m** utilise une police différente, plus élégante, presque calligraphique.

Je ne connais pas le nom de la police que **m** utilise, et donc dans la liste ci-dessus je n'ai pas pu représenter exactement le type de chiffres que cette option génère. Je suggère aux lecteurs de la tester par eux-mêmes.

12.3.2 Saisie des éléments d'une liste

En règle générale, les éléments d'une liste créée avec `\startitemize` sont saisis avec la commande `\item` qui possède également une version sous forme d'environnement plus adaptée au style Mark IV : `\startitem ... \stopitem`. Ainsi, observez l'exemple suivant :

```
\startitemize[a, packed]
\startitem Premier élément
\stopitem
\startitem Second élément \stopitem
\startitem Troisième élément
\stopitem
\stopitemize

\startitemize[a, packed]
\item Premier élément
\item Second élément
\item Troisième élément
\stopitemize
```

- a. Premier élément
 - b. Second élément
 - c. Troisième élément
- a. Premier élément
 - b. Second élément
 - c. Troisième élément

`\item` ou `\startitem` est la commande *générale* pour insérer un élément dans la liste. Elle s'accompagne des commandes supplémentaires suivantes pour obtenir un résultat particulier :

\head Cette commande doit être utilisée à la place de `\item` lorsque l'on veut éviter d'insérer un saut de page après l'élément en question. Son formatage peut être indiqué avec l'option `headstyle=` (`bold` par exemple).

Une construction courante consiste à inclure une liste imbriquée ou un bloc de texte immédiatement sous un élément de liste, de sorte que l'élément de liste fonctionne, en un sens, comme le *titre* de la sous-liste ou du bloc de texte. Dans ce cas, il est déconseillé d'insérer un saut de page entre cet élément et les paragraphes suivants. Si nous utilisons `\head` au lieu de `\item` pour saisir ces éléments, ConTeXt s'efforcera (dans la mesure du possible) de ne pas séparer cet élément du bloc suivant.

\nop annule l'affichage du séparateur.

\sym La commande `\sym{Texte}` permet de saisir un élément dans lequel le texte utilisé comme argument de `\sym` est utilisé comme *séparateur*, et non comme nombre ou symbole. Cette liste, par exemple, est construite avec des éléments saisis au moyen de `\sym` au lieu de `\item`.

```
\startitemize[a, packed]
\item Premier élément
\sym{x} Second élément
\nop Troisième élément
\stopitemize
```

a. Premier élément
x Second élément
Troisième élément

\sub Cette commande, qui ne doit être utilisée que dans les listes ordonnées (où chaque élément est précédé d'un numéro ou d'une lettre dans l'ordre alphabétique), fait en sorte que l'élément saisi avec elle conserve le numéro de l'élément précédent et, afin d'indiquer que le numéro est répété et qu'il ne s'agit pas d'une erreur, le signe « + » est imprimé à gauche. Cela peut être utile si l'on se réfère à une liste antérieure pour laquelle on suggère des modifications mais où, pour des raisons de clarté, il convient de conserver la numérotation de la liste originale.

\mar Cette commande conserve la numérotation des éléments, mais ajoute une lettre ou un caractère dans la marge (qui lui est passé en argument, entre crochets). Je ne suis pas sûr de son utilité.

```
\startnarrower[left]
\startitemize[n, packed]
\item Premier élément
\mar{o} Second élément (un o en
marge)
\sub Second élément
\item Troisième élément
\stopitemize
\stopnarrower
```

1. Premier élément
2. Second élément (un o en marge)
+ 2. Second élément
3. Troisième élément

Il existe deux commandes supplémentaires pour la saisie d'éléments, dont la combinaison produit des effets très *intéressants* et, si je peux me permettre, je pense qu'il est préférable de les expliquer par un exemple. `\ran` (abréviation de *gamme*) et `\its`, abréviation de *items*. La première prend un argument (entre crochets) et la seconde répète le symbole utilisé comme séparateur dans la liste un nombre x de fois (par défaut 4 fois, mais nous pouvons modifier cela en utilisant l'option `items`). L'exemple suivant montre comment ces deux commandes peuvent fonctionner ensemble pour créer une liste qui imite un questionnaire :

Après avoir lu l'introduction suivante, répondez aux questions suivantes~:

```
\startitemize[5, packed] [width=8em, distance=2em, items=5]
\ran{No \hss Yes} % \hss = un espace infiniment extensible et écrasable
\its Je n'utiliserais jamais ConTeXt, c'est trop difficile.
\its Je ne l'utiliserais que pour écrire de gros livres.
\its Je l'utiliserais toujours.
\its Je l'aime tellement que j'appellerais mon prochain enfant \quotation{Hans}, en hommage à Hans Hagen.
\stopitemize
```

Après avoir lu l'introduction suivante, répondez aux questions suivantes :

No	Yes	
◦ ◦ ◦ ◦ ◦	Je n'utiliserais jamais ConTeXt, c'est trop difficile.	
◦ ◦ ◦ ◦ ◦	Je ne l'utiliserais que pour écrire de gros livres.	
◦ ◦ ◦ ◦ ◦	Je l'utiliserais toujours.	
◦ ◦ ◦ ◦ ◦	Je l'aime tellement que j'appellerais mon prochain enfant "Hans", en hommage à Hans Hagen.	

12.3.3 Configuration basique des listes

Nous rappelons que « `itemize` » permet deux arguments. Nous avons déjà vu comment le premier argument nous permet de sélectionner le type de liste que nous voulons. Mais nous pouvons également l'utiliser pour indiquer d'autres caractéristiques de la liste ; ceci est fait par les options suivantes pour « `itemize` » dans son premier argument :

- `columns` : cette option détermine que la liste est composée de deux colonnes ou plus. Après l'option `colonnes`, le nombre de colonnes souhaité doit être écrit sous forme de mots séparés par une virgule : `two`, `three`, `four`, `five`, `six`, `seven`, `eight` or `nine`. `columns` non suivi d'un nombre quelconque génère deux colonnes.
- `intro` : cette option tente de ne pas séparer la liste, par un saut de ligne, du paragraphe qui la précède.
- `continue` : dans les listes ordonnées (numériques ou alphabétiques), cette option permet à la liste de poursuivre la numérotation à partir de la dernière liste numérotée. Si l'option `continue` est utilisée, il n'est pas nécessaire d'indiquer le type de liste que l'on souhaite, car on suppose qu'elle sera identique à la dernière liste numérotée.
- `packed` : est l'une des options les plus utilisées. Son utilisation permet de réduire au maximum l'espace vertical entre les différents *items* de la liste.
- `nowhite` produit un effet similaire à celui de `packed`, mais plus radical : il réduit non seulement l'espace vertical entre les éléments, mais aussi l'espace vertical entre la liste et le texte environnant.

```

Texte introductif.
\startitemize[packed]
\item Premier élément
\item Second élément
\item Troisième élément
\stopitemize
Texte introductif.
\startitemize[nowhite]
\item Premier élément
\item Second élément
\item Troisième élément
\stopitemize
Texte introductif.

```

Texte introductif.

- Premier élément
- Second élément
- Troisième élément

Texte introductif.

- Premier élément
- Second élément
- Troisième élément

Texte introductif.

- `after`, `before` : rajoutent au contraire des espaces respectivement après ou avant la liste

```

Texte introductif.
\startitemize[nowhite,after]
\item Premier élément
\item Second élément
\item Troisième élément
\stopitemize
Texte introductif.
\startitemize[nowhite,before]
\item Premier élément
\item Second élément
\item Troisième élément
\stopitemize
Texte introductif.

```

Texte introductif.

- Premier élément
- Second élément
- Troisième élément

Texte introductif.

- Premier élément
- Second élément
- Troisième élément

Texte introductif.

- `joinedup` : assure le rôle de `nowhite` mais entre un niveau d'`item` et son parent dans le cas de listes imbriquées.

```

\setupitemgroup[itemize] [1] [nowhite]%
\setupitemgroup[itemize] [2] [nowhite]%
\startitemize
\item item 1.1
\startitemize
\item item 2.1 \item item 2.2
\stopitemize
\item item 1.2
\stopitemize

\setupitemgroup[itemize] [1] [joinedup]%
\startitemize
\item item 1.1
\startitemize
\item item 2.1 \item item 2.2
\stopitemize
\item item 1.2
\stopitemize

```

- item 1.1
 - item 2.1
 - item 2.2
- item 1.2
 - item 1.1
 - item 2.1
 - item 2.2
 - item 1.2

- **broad** : augmente l'espace horizontal entre le séparateur d'élément et le texte de l'élément. L'espace peut être augmenté en multipliant un nombre par **broad** comme dans, par exemple : `\startitemize[2*broad]`.

```
\startitemize[packed,2*broad]
\item Premier élément
\item Second élément
\stopitemize
```

- Premier élément
- Second élément

- **serried** : supprime l'espace horizontal entre le séparateur d'élément et le texte.
- **intext** : supprime le retrait suspendu.
- **text** : supprime le retrait suspendu et réduit l'espace vertical entre les éléments.
- **repeat** : dans les listes imbriquées, la numérotation d'un niveau enfant *repeat* devient le même niveau que le niveau précédent. Ainsi, nous aurions, au premier niveau : 1, 2, 3, 4 ; au deuxième niveau : 1.1, 1.2, 1.3, etc. L'option doit être indiquée pour la liste intérieure, pas pour la liste extérieure.
- **margin**, **inmargin** : par défaut, le séparateur de liste est imprimé à gauche, mais à l'intérieur de la zone de texte elle-même (**atmargin**). Les options **margin** et **inmargin** permettent de déplacer le séparateur vers la marge.

12.3.4 Configuration complémentaire des listes

Le deuxième argument, également facultatif, de `\startitemize` permet une configuration plus détaillée et plus approfondie des listes.

- **before**, **after** : commandes à exécuter respectivement avant le démarrage ou après la fermeture de l'environnement **itemize** (à ne pas confondre avec ces options de l'argument 1, qui font référence aux espacements verticaux voir ci-dessus).
- **inbetween** : commande à exécuter entre deux **items**.
- **beforehead**, **afterhead** : commande à exécuter avant ou après un élément saisi avec la commande `\head`.
- **left**, **right** : caractère à imprimer à gauche ou à droite du séparateur. Par exemple, pour obtenir des listes alphabétiques dans lesquelles les lettres sont entourées de parenthèses, il faudrait écrire `\startitemize[a][left=(), right=)]`
- **stopper** : indique un caractère à écrire après le séparateur. Ne fonctionne que dans les listes ordonnées.
- **width**, **maxwidth** : largeur de l'espace réservé au séparateur et donc au retrait suspendu.
- **factor** : nombre représentatif du facteur de séparation entre le séparateur et le texte.
- **distance** : mesure de la distance entre le séparateur et le texte.

```

\setupitemgroup[itemize] [1]
    [nowhite,joinedup,3*broad]%
\setupitemgroup[itemize] [2]
    [nowhite] [width=5mm]%
\startitemize
\item item 1.1
\startitemize
\item item 2.1 \item item 2.2
\stopitemize
\item item 1.2
\stopitemize
\setupitemgroup[itemize] [1] [distance=5mm]%
\setupitemgroup[itemize] [2] [distance=5mm]%
\startitemize
\item item 1.1
\startitemize
\item item 2.1 \item item 2.2
\stopitemize
\item item 1.2
\stopitemize

```

- item 1.1
 - item 2.1
 - item 2.2
- item 1.2
- item 1.1
 - item 2.1
 - item 2.2
- item 1.2

- `leftmargin`, `rightmargin`, `margin` : marge à ajouter à gauche (`leftmargin`) ou à droite (`rightmargin`) des éléments.
- `start` : numéro à partir duquel la numérotation des éléments commencera.
- `symalign`, `itemalign`, `align` : alignement des éléments. Permet les mêmes valeurs que `\setupalign`. `symalign` contrôle l'alignement du séparateur, `itemalign` celui du texte de l'élément et `align` contrôle les deux.
- `indenting` : indentation de la première ligne des paragraphes à l'intérieur de l'environnement. Voir section 11.1.1.
- `indentnext` : indique si le paragraphe suivant l'environnement doit être indenté ou non. Les valeurs sont *yes*, *no* et *auto*.
- `items` : dans les éléments saisis en entrée avec `\its`, indique le nombre de fois que le séparateur doit être reproduit.
- `style`, `color`; `headstyle`, `headcolor`; `marstyle`, `marcolor`; `symstyle`, `symcolor`: ces options contrôlent le style et la couleur des éléments lors de leur saisie dans l'environnement avec les commandes `\item`, `\head`, `\mar` ou `\sym`.

12.3.5 Listes simples à l'aide de la commande `\items`.

Une alternative à l'environnement `itemize` pour les listes non numérotées très simples, où les éléments ne sont pas trop gros, est la commande `\items` dont la syntaxe est :

```
\items[Configuration]{Item 1, Item 2, ..., item n}
```

Les différents éléments de la liste sont séparés les uns des autres par des virgules.
Par exemple :

Les fichiers graphiques peuvent avoir, entre autres, les extensions suivantes :
`\items{png, jpg, tiff, bmp}`

Les fichiers graphiques peuvent avoir, entre autres, les extensions suivantes :

- png
- jpg
- tiff
- bmp

Les options de configuration prises en charge par cette commande sont un sous-ensemble de celles de la commande `itemize`, à l'exception de deux options spécifiques à cette commande :

- `symbol` : cette option détermine le type de liste qui sera généré. Elle prend en charge les mêmes valeurs que celles utilisées pour `itemize` pour sélectionner un type de liste.
- `n` : cette option indique à partir de quel numéro d'élément il y aura un séparateur.

12.3.6 Prédétermination du comportement des listes et création de nos propres types de listes

Dans les sections précédentes, nous avons vu comment indiquer quel type de liste nous voulons et quelles caractéristiques elle doit avoir. Mais faire cela à chaque fois qu'une liste est appelée est inefficace et produira généralement un document incohérent dans lequel chaque liste a sa propre apparence, mais sans que les différentes apparences ne répondent à aucun critère.

Résultat préférable pour cela :

- Prédéterminer le comportement *normal* de `itemize` et `\items` dans le préambule du document.
- Créer nos propres listes personnalisées. Par exemple : une liste numérotée alphabétiquement que nous voulons appeler *ListAlpha*, une liste numérotée avec des chiffres romains (*ListRoman*), etc.

Nous réalisons la première avec les commandes `\setupitemize` et `\setupitems`. La seconde nécessite l'utilisation de la commande `\defineitemgroup`, ou `\defineitems`. La première créera un environnement de liste similaire à `itemize` et la seconde une commande similaire à `items`.

12.4 Descriptions et énumérations

Les descriptions et les énumérations sont deux constructions qui permettent la composition cohérente de paragraphes ou de groupes de paragraphes qui développent, décrivent ou définissent une phrase ou un mot.

12.4.1 Descriptions

Pour les descriptions, nous faisons la différence entre un *titre* et son explication ou développement. Nous pouvons créer une nouvelle description avec :

```
\definedescription [^1] [^2] [...] [^3=...]
1  NAME
2  NAME
3  inherits: \setupdescription
```

où le premier *Name* est le nom sous lequel cette nouvelle construction sera connue, le second correspond à un environnement de description déjà existant et dont le nouveau va hériter, et le dernier arguments contrôle ce à quoi notre nouvel environnement ressemblera. Après la déclaration précédente, nous aurons une nouvelle commande et un environnement portant le nom que nous avons choisi. Ainsi :

```
\definedescription [Concept]

\Concept{Protestation} Il vient une heure
où protester ne suffit plus : après la
philosophie, il faut l'action.

Ceci est une citation de Victor Hugo.

\startConcept{Rire}
Faire rire, c'est faire oublier.

Ceci est une citation de Victor Hugo.
\stopConcept
```

Protestation Il vient une heure où protester ne suffit plus : après la philosophie, il faut l'action.

Ceci est une citation de Victor Hugo.

Rire Faire rire, c'est faire oublier.
Ceci est une citation de Victor Hugo.

Nous pouvons utiliser la commande pour le cas où le texte explicatif du titre ne comporte qu'un seul paragraphe, mais généralement nous utiliserons plutôt l'environnement car il permet de traiter le cas plus général où la description occupe plus d'un paragraphe, contient des flottants etc. Lorsque la commande est utilisée, seul le paragraphe qui la suit immédiatement est celui qui sera considéré comme le texte explicatif du titre. Lorsque l'environnement est utilisé, tout le contenu sera formaté avec l'indentation appropriée pour faire apparaître clairement son lien avec le titre.

```
\definedescription
[Concept]
[alternative=left, width=1cm, headstyle=bold]
\startConcept{Contextualiser}
Placer quelque chose dans un certain contexte,
ou composer un texte avec le système de
composition appelé \ConTeXt. La capacité
à contextualiser correctement dans toute
situation est considérée comme un signe
d'intelligence et de bon sens.
\stopConcept
```

Contextualiser Placer quelque chose dans un certain contexte, ou composer un texte avec le système de composition appelé ConTeXt. La capacité à contextualiser correctement dans toute situation est considérée comme un signe d'intelligence et de bon sens.

Comme c'est normalement le cas avec ConTeXt, l'aspect qu'aura notre nouvelle construction peut être indiqué au moment de sa création, avec le dernier argument ou plus tard avec `\setupdescription`:

```
\setupdescription [...] [...]
OPT
1 NAME
2 title      = yes no
level       = NUMBER
text        = TEXT
headcommand = \...##1
before      = COMMAND
after       = COMMAND
inbetween   = COMMAND
alternative = left right inmargin inleft inright margin leftmargin
               rightmargin innermargin outermargin serried hanging
               top empty command NAME
align       = inherits: \setupalign
headalign   = inherits: \setupalign
indenting   = inherits: \setupindenting
display     = yes no
indentnext  = yes no auto
width       = fit broad line DIMENSION
distance    = none DIMENSION
stretch     = NUMBER
shrink      = NUMBER
hang        = fit broad none margin NUMBER
closesymbol = COMMAND
closecommand = \...##1
expansion   = yes no xml
referenceprefix = + - TEXT
sample      = TEXT
margin      = yes no standard DIMENSION
style       = STYLE COMMAND
color       = COLOR
headstyle   = STYLE COMMAND
headcolor   = COLOR
aligntitle  = yes no
```

où 1 est le nom (ou la liste de nom) de notre nouvelle description, et 2 détermine à quoi elle ressemble. Parmi les différentes options de configuration possibles, je soulignerai :

- **alternative** cette option est celle qui contrôle fondamentalement l'apparence de la construction. Elle détermine le placement du titre par rapport à sa description. Ses valeurs possibles sont les suivantes : `left`, `right`, `inmargin`, `inleft`, `inright`, `margin`, `leftmargin`, `rightmargin`, `innermargin`, `outermargin`, `serried`, `hanging`, leurs noms sont suffisamment clairs pour se faire une idée du résultat, même si, en cas de doute, il est préférable de faire un test pour voir ce que cela donne.
- **width** contrôle la largeur de la boîte dans laquelle le titre sera écrit. Selon la valeur de `alternative`, cette distance fera également partie de l'indentation avec laquelle le texte explicatif sera écrit.
- **distance** contrôle la distance entre le titre et l'explication.
- **headstyle**, **headcolor**, **headcommand** affecte l'aspect du titre lui-même : Style (`headstyle`) et couleur (`headcolor`). Avec `headcommand`, on peut indiquer une commande à laquelle le texte du titre sera passé comme argument. Par exemple : `headcommand=\WORD` fera en sorte que le texte du titre soit tout en majuscules.
- **style**, **color** contrôle l'apparence du texte descriptif du titre.
- **hang** contrôle les spécificités dans le cas de l'alternative `hanging` (combien de ligne sont mise en retrait).

```
\startbuffer
Il vient une heure où protester ne suffit
plus : après la philosophie, il faut l'action.
\stopbuffer%
\definedescription [Concept]
[alternative=left,
 width=2.5cm,
 distance=2em,
 headstyle=\tt\bf,
 headcolor=darkred]%
\startConcept{left} \getbuffer \stopConcept

\definedescription [Concept] [alternative=hanging]
\startConcept{hanging} \getbuffer \stopConcept
\definedescription [Concept] [alternative=top]
\startConcept{top} \getbuffer \stopConcept
\definedescription [Concept] [alternative=serried]
\startConcept{serried} \getbuffer \stopConcept
\definedescription [Concept] [distance=5em]
\startConcept{distance} \getbuffer \stopConcept
\definedescription [Concept] [alternative=right,
distance=2em]
\startConcept{right} \getbuffer \stopConcept
\definedescription [Concept] [headalign=flushright]
\startConcept{headalign} \getbuffer \stopConcept
```

left

Il vient une heure où protester ne suffit plus : après la philosophie, il faut l'action.

hanging

Il vient une heure où protester ne suffit plus : après la philosophie, il faut l'action.

top

Il vient une heure où protester ne suffit plus : après la philosophie, il faut l'action.

serried

Il vient une heure où protester ne suffit plus : après la philosophie, il faut l'action.

distance

Il vient une heure où protester ne suffit plus : après la philosophie, il faut l'action.

Il vient une heure où protester ne suffit plus : après la philosophie, il faut l'action.

headalign

Il vient une heure où protester ne suffit plus : après la philosophie, il faut l'action.

12.4.2 Énumérations

Les énumérations sont des éléments de texte numérotés et structurés sur plusieurs niveaux. Chaque élément commence par un titre qui se compose, par défaut, du nom de la structure et de son numéro, bien que nous puissions modifier le titre avec l'option `text` (voir le second exemple page suivante). Elles sont assez similaires aux descriptions, mais offrent deux distinctions :

- Tous les éléments d'une énumération partagent le même titre.
- Ils diffèrent donc les uns des autres par leur numérotation.

Cet environnement peut être très utile, par exemple, pour rédiger des énoncés, des définitions, des formules, des problèmes ou des exercices dans un manuel, en veillant à ce qu'ils soient correctement numérotés et formatés de manière cohérente et automatique.

```
\defineenumeration [^1] [^2] [^3=...]
1 NAME
2 NAME
3 inherits: \setupenumeration
```

```
\defineenumeration [Exercice]
[alternative=top,
 before=\blank,
 after=\blank,
 between=\blank]

\Exercice
pas facile celui-ci A.\par
pas facile celui-ci B.

\startExercice
Commençons par le commencement A.\par
Commençons par le commencement B.

\startsubExercice
Celui ci est élémentaire A.\par
Celui ci est élémentaire B.
\stopsubExercice

\startsubExercice
Celui ci est un complément A.\par
Celui ci est un complément B.
\stopsubExercice

\subExercice
pour finir A.\par
pour finir B.

\stopExercice
```

Exercice 1

pas facile celui-ci A.

pas facile celui-ci B.

Exercice 2

Commençons par le commencement A.
Commençons par le commencement B.

subExercice 2.1

Celui ci est élémentaire A.
Celui ci est élémentaire B.

subExercice 2.2

Celui ci est un complément A.
Celui ci est un complément B.

subExercice 2.3

pour finir A.

pour finir B.

Ainsi, dans l'exemple précédent, on constate que `\defineenumeration` génère automatiquement une série de nouvelles commandes associées au nom du nouvel environnement (`\startExercice`, `\startsubExercice`, ...), en effet les énumérations peuvent avoir jusqu'à quatre niveaux de profondeur. Tout comme pour les descriptions, nous utiliserons généralement l'environnement `\startNom ... \stopNom` qui permet de couvrir plusieurs paragraphes, mais les commandes simples (`\Exercice`, `\subExercice`, ...) peuvent également être utilisées dans le cas de paragraphe seul.

```
\defineenumeration [Exercice]
  [alternative=left,
  headcolor=darkcyan,
  width=2cm,
  text={Exercice},
  before=\blank,
  after=\blank,
  between=\blank]

\startExercice
Commençons par le commencement A.\par
Commençons par le commencement B.

\startsubExercice
Celui ci est élémentaire A.\par
Celui ci est élémentaire B.
\stopsubExercice

\startsubExercice
Celui ci est un complément A.\par
Celui ci est un complément B.
\stopsubExercice

\stopExercice
```

Exercice 1	Commençons par le commencement A. Commençons par le commencement B.
Exercice 1.1	Celui ci est élémentaire A. Celui ci est élémentaire B.
Exercice 1.2	Celui ci est un complément A. Celui ci est un complément B.

L'apparence des énumérations (voir exemple ci-dessus) peut être déterminée au moment de leur création ou ultérieurement avec `\setupenumeration` dont les options et valeurs sont similaires à celles de `\setupdescription`.

Pour chaque énumération, nous pouvons configurer chacun de ses niveaux séparément. Ainsi, par exemple, `\setupenumeration [subExercice] [Configuration]` affectera le deuxième niveau de l'énumération appelée « Exercice ».

```

\setupenumeration [...] [...]
OPT

1 NAME
2 title      = yes no
number      = yes no
numbercommand = \...##1
titledistance = DIMENSION
titlestyle   = STYLE COMMAND
titlecolor   = COLOR
titlecommand = \...##1
titleleft    = COMMAND
titleright   = COMMAND
left         = COMMAND
right        = COMMAND
symbol       = COMMAND
starter      = COMMAND
stopper      = COMMAND
coupling     = NAME
counter      = NAME
level        = NUMBER
text         = TEXT
headcommand  = \...##1
before       = COMMAND
after        = COMMAND
inbetween    = COMMAND
alternative  = left right inmargin inleft inright margin leftmargin
               rightmargin innermargin outermargin serried hanging
               top empty command NAME
align        = inherits: \setupalign
headalign    = inherits: \setupalign
indenting    = inherits: \setupindenting
display      = yes no
indentnext   = yes no auto
width        = fit broad line DIMENSION
distance     = none DIMENSION
stretch      = NUMBER
shrink       = NUMBER
hang         = fit broad none margin NUMBER
closesymbol  = COMMAND
closecommand = \...##1
expansion    = yes no xml
referenceprefix = + - TEXT
sample       = TEXT
margin       = yes no standard DIMENSION
style        = STYLE COMMAND
color        = COLOR
headstyle    = STYLE COMMAND
headcolor    = COLOR
aligntitle   = yes no
inherits: \setupcounter

```

Pour contrôler la numérotation, il existe les commandes supplémentaires suivantes :

- **\setEnumerationName** définit la valeur de numérotation actuelle.
- **\resetEnumerationName** remet le compteur d'énumération à zéro.
- **\nextEnumerationName** augmente le compteur d'énumération d'une unité.

12.5 Lignes et cadres

Il est dit dans le manuel de référence de ConTEXt que TeX a une énorme capacité de gestion du texte, mais est très faible dans la gestion des informations graphiques. Je ne suis pas d'accord : il est vrai que pour la gestion des lignes et des cadres, les possibilités de ConTEXt (en fait TeX) ne sont pas aussi écrasantes que lorsqu'il s'agit de composer du texte. Mais dire que le système est faible à cet égard est, je pense, un peu exagéré. Je ne connais aucune fonction avec des lignes et des cadres que d'autres systèmes de composition peuvent faire pour des documents et que ConTEXt est incapable de générer. Et si nous combinons ConTEXt avec MetaPost, ou avec TiKZ (ConTEXt a un module d'extension pour cela), alors les possibilités ne sont limitées que par notre imagination.

Dans les sections suivantes, cependant, je me limiterai à expliquer comment générer de simples lignes horizontales et verticales et des cadres autour des mots, des phrases ou des paragraphes.

12.5.1 Lignes simples

La façon la plus simple de dessiner une ligne horizontale est d'utiliser la commande `\hairline` qui génère une ligne horizontale occupant toute la largeur d'une ligne de texte normale.

Il ne peut y avoir de texte d'aucune sorte sur la ligne où se trouve la ligne générée par `\hairline`. Afin de générer une ligne capable de coexister avec le texte sur la même ligne, nous avons besoin de la commande `\thinrule`. Cette deuxième commande utilisera toute la largeur de la ligne. Par conséquent, dans un paragraphe isolé, elle aura le même effet que `\hairline`, tandis que dans le cas contraire, `\thinrule` produira la même expansion horizontale que `\hfill` (voir section 10.3.3), mais au lieu de remplir l'espace horizontal avec un espace blanc (comme le fait `\hfill`), elle le remplit avec une ligne.

```
Avec hairline~:\par
Sur la gauche\hairline\\
\hairline sur la droite\\
Des deux \hairline côtés\\
\hairline centré\hairline
\blank[big]

Avec thinrule~:\par
Sur la gauche\thinrule\\
\thinrule sur la droite\\
Des deux \thinrule côtés\\
\thinrule centré\thinrule
```

Avec hairline :	Sur la gauche

	sur la droite
	Des deux

	côtés

	centré

Avec thinrule :	Sur la gauche

	sur la droite
	Des deux

	côtés

	centré

La commande `\thinrules` permet de générer plusieurs lignes. Par exemple, `\thinrules[n=2]` générera deux lignes consécutives, chacune de la largeur de la ligne normale. Les lignes générées avec `\thinrules` peuvent également être configurées, soit dans un appel direct à la commande, en indiquant la configuration comme l'un de ses arguments, soit de manière générale avec `\setupthinrules`. La configuration comprend l'épaisseur de la ligne (`rulethickness`), sa couleur (`color`), la couleur de fond (`background`), l'espace interligne (`interlinespace`), etc.

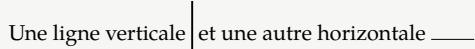
```
\setupthinrules[color=darkyellow,height=-0.5mm,depth=1.0mm]  
Sur la gauche\thinrules[n=3] centre décalé \thinrule sur la droite
```



Je laisserai un certain nombre d'options sans explication. Le lecteur peut les consulter dans `setup-fr.pdf` (voir section 3.6). Certaines options ne diffèrent des autres qu'en termes de nuance (c'est-à-dire qu'il n'y a pratiquement aucune différence entre elles), et je pense qu'il est plus rapide pour le lecteur d'essayer de voir la différence, que pour moi d'essayer de la transmettre avec des mots. Par exemple : l'épaisseur de la ligne que je viens de dire dépend de l'option `rulethickness`. Mais elle est également affectée par les options `height` et `depth` (voir l'exemple ci-dessus).

Des lignes plus petites peuvent être générées avec les commandes `\hl` et `\vl`. La première génère une ligne horizontale et la seconde une ligne verticale. Toutes deux prennent comme paramètre un nombre qui nous permet de calculer la longueur de la ligne. Dans `\hl`, le nombre mesure la longueur en *ems* (il n'est pas nécessaire d'indiquer l'unité de mesure dans la commande) et dans `\vl`, l'argument fait référence à la hauteur actuelle de la ligne.

```
Une ligne verticale \vl[2] et une autre horizontale \hl[2]
```



Ainsi, `\hl[2]` génère une ligne horizontale de 3 ems et `\vl[2]` génère une ligne verticale de la hauteur correspondant à trois lignes. N'oubliez pas que l'indicateur de mesure de ligne doit être inséré entre crochets, et non entre accolades. Dans les deux commandes, l'argument est facultatif. S'il n'est pas saisi, la valeur 1 est prise en compte.

`\fillinline` est une autre commande (je pense dépréciée, voyez ensuite `\fillinrules`) permettant de créer des lignes horizontales de longueur précise. Elle prend en charge davantage de configuration dans laquelle nous pouvons indiquer (ou pré-déterminer avec `\setupfillinlines`) la largeur (option `width`) en plus de quelques autres caractéristiques.

Une particularité de cette commande est que le texte qui est écrit en argument sera placé à gauche de la ligne, séparant ce texte de la ligne par l'espace blanc nécessaire pour occuper toute la ligne. Attention, il faut bien penser à indiquer un changement de paragraphe entre deux lignes de ce type. Par exemple :

```
\fillinline[width=2cm]{Nom} \par
\fillinline[width=4cm]{Prénom}

\fillinline[width=4cm]{Adresse} \par
```

Nom	<hr/>
Prénom	<hr/>
Adresse	<hr/>

Outre la largeur de la ligne, nous pouvons configurer la marge (`margin`), la distance (`distance`), l'épaisseur (`rulethickness`) et la couleur (`color`).

`\fillinrules` a un rôle très proche de `\fillinline` mais plus générale car elle permet d'insérer plus d'une ligne (option « `n` ») et traite automatiquement le changement de ligne et de paragraphe.

```
\setupfillinrules[separator=:,interlinespace=small]
\fillinrules[n=1]{Nom}{(en majuscule)}
\fillinrules[n=1,width=2cm]{Prénom}
\fillinrules[n=3,width=fit]{Adresse 1}
\fillinrules[n=3,distance=1cm]{Adresse 2}
```

Nom:	<hr/> <hr/> <hr/>	(en majuscule)
Prénom :	<hr/> <hr/> <hr/>	
Adresse 1:	<hr/> <hr/> <hr/>	
Adresse 2:	<hr/> <hr/> <hr/>	

Dernier type de ligne : `\blackrule`. Voyez la comparaison avec `thinrule` :

```
\setupthinrules[color=darkyellow,height=-0.5mm,depth=1.0mm]
Sur la gauche\thinrules[n=3] centre décalé \thinrule sur la droite

\setupblackrules[color=darkgreen,height=-0.5mm,depth=1.0mm]
Sur la gauche\blackrules[n=3] centre décalé \blackrule sur la droite
```



Contrairement à `\thinrule`, `\blackrule` produit des lignes de longeur fixe, que l'on définit avec l'option `width`.

Un manuel spécifique aux lignes a été produit en 2018 intitulé [Rules](#). N'hésitez pas à le consulter.

12.5.2 Lignes liées au texte

Bien que certaines des commandes que nous venons de voir puissent générer des lignes qui coexistent avec du texte sur la même ligne, ces commandes se concentrent en fait sur la mise en page de la ligne. Pour écrire des lignes liées à un certain texte, ConTeXt a des commandes :

- qui génèrent du texte entre les lignes.
- qui génèrent des lignes sous le texte (soulignement, underlining), au-dessus du texte (surlignage, overlining) ou à travers le texte (barré, strikethrough).

Pour générer un texte entre les lignes, la commande habituelle est `\textrule`. Cette commande trace une ligne qui traverse toute la largeur de la page et écrit le texte qu'elle prend comme paramètre sur le côté gauche (mais pas dans la marge). Par exemple :

```
Texte avant.  
\textrule[Texte en exemple]  
Texte encore après.
```

Texte avant.
— **Texte en exemple** —————
Texte encore après.

`\textrule` permet un premier argument facultatif avec trois valeurs possibles : `top`, `middle` et `bottom` selon la position souhaitée par rapport au reste du texte.

```
Texte A.  
\textrule[top]{Texte en exemple}  
Texte B.  
\textrule[middle]{Texte en exemple}  
Texte C.  
\textrule[bottom]{Texte en exemple}  
Texte D.
```

Texte A.

— **Texte en exemple** —————
Texte B.

— **Texte en exemple** —————
Texte C.

— **Texte en exemple** —————

Texte D.

Similaire à `\textrule`, l'environnement `\starttextrule` permet d'insérer une ligne de texte au début de l'environnement, mais aussi une ligne horizontale à la fin.

```
Texte avant.  
\starttextrule{Texte en exemple}  
Texte utilisé en contenu de  
l'environnement, pour faire joli.  
\stoptextrule  
Texte encore après.
```

```
Texte avant.  
— Texte en exemple —————  
Texte utilisé en contenu de l'environnement,  
pour faire joli.  
——————  
Texte encore après.
```

\textrule et starttextrule peuvent être configurés avec \setuptextrules.

```
\startbuffer  
Texte utilisé en contenu de l'environnement, pour faire joli.  
\stopbuffer  
  
\setuptextrules[location=left] % sinon inmargin  
\starttextrule{location=left} \getbuffer \stoptextrule  
  
\setuptextrules[width=2cm] % largeur du trait à gauche  
\starttextrule{width} \getbuffer\stoptextrule  
  
\setuptextrules[distance=2em] % distance entre le trait et le texte  
\starttextrule{distance} \getbuffer\stoptextrule  
  
\setuptextrules[style=\bfa,color=darkgreen,rulecolor=darkred]  
\starttextrule{style et couleur} \getbuffer\stoptextrule
```

— location=left —————
Texte utilisé en contenu de l'environnement, pour faire joli.
——————

— width —————
Texte utilisé en contenu de l'environnement, pour faire joli.
——————

— distance —————
Texte utilisé en contenu de l'environnement, pour faire joli.
——————

— style et couleur —————
Texte utilisé en contenu de l'environnement, pour faire joli.
——————

Pour tracer des lignes sous, sur ou à travers du texte, les commandes suivantes sont utilisées :

```
\setupinterlinespace[big]
\underbar{Ceci est un texte underbar} \\
\underbar{Ceci \underbar{est \underbar{un}} texte}} underbar} \\
\underbars{Ceci est un texte underbars} \\
\overbar{Ceci est un texte overbar} \\
\overbars{Ceci est un texte overbars} \\
\overstrike{Ceci est un texte overstrike} \\
\overstrikes{Ceci est un texte overstrikes}
```

Ceci est un texte underbar
Ceci est un texte underbar
Ceci est un texte underbars
Ceci est un texte overbar
Ceci est un texte overbars
Ceci est un texte overstrike
Ceci est un texte overstrikes

Comme on peut le voir, il existe deux commandes pour chaque type de ligne (sous, sur ou à travers le texte). La version singulière de la commande trace une seule ligne sous, sur ou à travers tout le texte pris comme argument, tandis que la version plurielle de la commande ne trace la ligne que sur les mots, mais pas sur les espaces blancs.

Ces commandes ne sont pas compatibles entre elles, c'est-à-dire qu'on ne peut pas en appliquer deux au même texte. Si on essaie, c'est toujours la dernière qui l'emportera. En revanche, `\underline` peut être imbriqué, soulignant ce qui a déjà été souligné.

Le manuel de référence signale que `\underline` désactive la césure des mots du texte qui constituent son argument. Il n'est pas clair pour moi si cette déclaration se réfère uniquement à `\underline` ou aux six commandes que nous examinons.

12.5.3 Mots ou textes encadrés

Les cadres sont un élément essentiel de ConTeXt. Ils permettent d'obtenir toute sorte de personnalisation et d'effets graphiques sur lesquels nous reviendrons, mais surtout leur utilisation est omniprésente dans quantité d'éléments de ConTeXt et donc bien les comprendre ainsi que l'effet des différentes options est très prévieux.

Pour entourer un texte d'un cadre ou d'une grille, on utilise :

- Les commandes `\framed` ou `\inframed` si le texte est relativement bref et ne prend pas plus d'une ligne.
- L'environnement `\startframedtext` pour les textes plus longs.

La différence entre `\framed` et `\inframed` réside dans le point à partir duquel le cadre est dessiné. Dans le cas de `\framed`, le cadre est positionné sur la ligne de base, sur laquelle reposent les lettres. Avec `\inframed`, c'est le mot qui est positionnés sur la ligne de base.

```
\tfd
```

Voici deux \framed{cadres} bien \inframed{encadrés}.

```
\showboxes
```

Voici deux \framed{cadres} bien \inframed{encadrés}.

Voici deux cadres bien encadrés.

Voici deux cadres bien encadrés.

Les deux, peuvent être personnalisés avec `\setupframed`, et `\startframedtext` est personnalisé avec `\setupframedtext`. Les options de personnalisation des deux commandes sont assez similaires. Elles nous permettent d'indiquer les dimensions du cadre (`height`, `width`, `depth`), la forme des coins (`framecorner`), qui peut être `rectangular` ou ronde (`round`), la couleur du cadre (`framecolor`), l'épaisseur du trait (`framethickness`), l'alignement du contenu (`align`), la couleur du texte (`foregroundcolor`), la couleur de l'arrière-plan (`background` et `backgroundcolor`), etc.

Pour `\startframedtext`, il existe également une propriété apparemment étrange : `frame=off` qui fait que le cadre n'est pas dessiné (il est toujours présent, mais invisible). Cette propriété existe parce que, le cadre entourant un paragraphe étant indivisible, il est courant que le paragraphe entier soit enfermé dans un environnement `framedtext` avec l'option de dessin du cadre désactivée, afin de s'assurer qu'aucun saut de page n'est inséré dans un paragraphe.

Nous pouvons également créer une version personnalisée de ces commandes avec `\defineframed` et `\defineframedtext`.

A. Premiers exemples

Nous allons voir quelques exemples, simples pour commencer.

```
\defineframed
[MonCadre]
[background=color,
 backgroundcolor=darkred,
 foregroundcolor=white,
 foregroundstyle=\bf]
A
\MonCadre{coucou 1}\hfill
\MonCadre[offset=2mm]{offset}\hfill
\MonCadre{coucou 2}\hfill
\MonCadre[frameoffset=2mm]{frameoffset}\hfill % conserve la position du mot
\MonCadre[backgroundoffset=2mm]{backgroundoffset}
```

A coucou 1

offset

coucou 2

frameoffset

backgroundoffset

Il y a ensuite tout une série de personnalisation sur le trait du cadre, sur la distance entre le texte et le cadre (en haut, en bas, à gauche, à droite), à découvrir au fur et à mesure de vos besoins.

```
\defineframed[MonTitre]
[frame=off, topframe=on,
leftframe=on,
rulethickness=2pt,
foregroundstyle=\bfa,
framecolor=darkred]%
\MonTitre{Superbe titre 1}
\blank
\MonTitre
[loffset=1em,toffset=2pt]
{Superbe titre 2}
\blank
\MonTitre
[corner=16,frame=on,offset=2pt]
{Superbe titre 3}
```

Superbe titre 1

Superbe titre 2

Superbe titre 3

B. Retour à la ligne et alignement

Attention, par défaut, le cadre ne fait aucun retour à la ligne. Il faut indiquer une valeur à `align` (voir [section 11.6.1](#) pour les valeurs à considérer, `normal` fait référence à un texte justifié, et [section 11.3.3](#) pour les options de tolérance). Dans l'exemple on utilise `align={normal,verytolerant}`.

```

\startbuffer
Si le texte est un peu trop grand,
par défaut il ne sera pas coupé
\stopbuffer%
\defineframed
[MonCadre]
[background=color,
 backgroundcolor=darkred,
 foregroundcolor=white,
 foregroundstyle=\bf]%
\MonCadre{\getbuffer} \blank
\MonCadre[width=5cm]{\getbuffer}
\blank
\MonCadre[width=5cm,align=normal]
{\getbuffer} \blank
\MonCadre[width=5cm,
           align={normal,verytolerant},
           backgroundcolor=darkgreen]
{\getbuffer}

```

Si le texte est un peu trop grand, par défaut il ne sera pas coupé

peu trop grand, par défaut il ne sera pas coupé

Si le texte est un peu trop grand,
par défaut il ne sera pas coupé

Si le texte est un peu trop
grand, par défaut il ne sera
pas coupé

C. Largeur

Un point concernant le paramètre `width`. La dimension de la largeur peut être indiquée directement (en cm, en pt, en fonction d'autre longueur par exemple « `0.5\textwidth` »).

Des dimensions automatiques sont proposées par ConTeXt. `broad` fait référence à la largeur totale disponible pour la zone de texte, `local` fait référence à la largeur totale disponible pour la zone en cours, et `fit` colle à la largeur du texte contenu par le cadre,

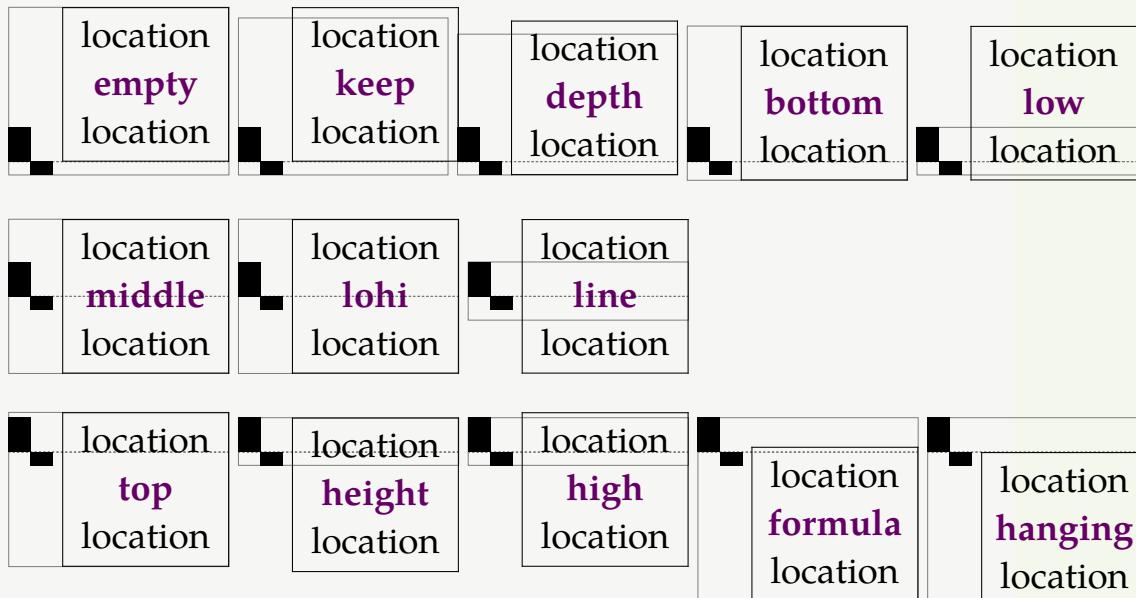
		1	
		width=6cm	
		width=0.5\textwidth	
		width=broad	
		width=local	
		width=fit	
		• width=broad	
		• width=local	
		• width=fit	



D. Positionnement par rapport à la ligne de base

```
\defineframed[MonCadre][width=2.25cm,align=middle]
\define[1]\DemoLoc{\rule{\fboxwidth}{0pt}\color{darkmagenta}{\bf #1}\rule{\fboxwidth}{0pt}}
{\getbuffer \MonCadre[location=#1]
 {location\\ \color{darkmagenta}{\bf #1}\\location}}}
\setupbodyfont[14pt]
%\showboxes
\startbuffer
\blackrule[height=max,depth=0pt,width=3mm]%
\blackrule[height=0pt,depth=max,width=3mm]
\stopbuffer

\strut
\DemoLoc{empty} \dontleavehmode \DemoLoc{keep} \dontleavehmode
\DemoLoc{depth} \dontleavehmode \DemoLoc{bottom} \dontleavehmode
\DemoLoc{low}
\blank[big]\strut
\DemoLoc{middle} \dontleavehmode \DemoLoc{lohi} \dontleavehmode
\DemoLoc{line}
\blank[big]\strut
\DemoLoc{top} \dontleavehmode \DemoLoc{height} \dontleavehmode
\DemoLoc{high} \dontleavehmode \DemoLoc{formula} \dontleavehmode
\DemoLoc{hanging}
```



Plus complexe, ci-dessus les différents cas possibles pour le paramètre `location`. Vous voyez que de nombreux cas sont possibles, afin d'aligner le haut ou le bas de la boîte sur la ligne de base du texte, ou bien encore, souvent plus utile, d'aligner la première ou la dernière ligne de base de la boîte avec la ligne de base du texte.

E. Attention au paramètre strut

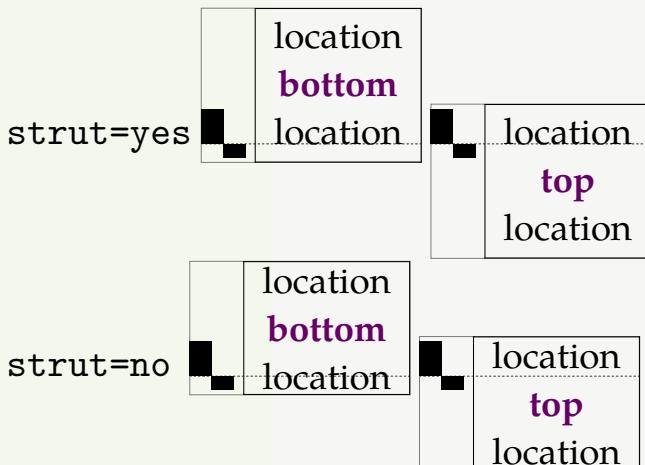
L'option `strut` force la première ligne de l'encadré à prendre tout la hauteur et toute la profondeur que la police en cours lui permet de prendre. Cela permet de « donner une hauteur de ligne standard ». Si vous avez des difficultés parfois à positionner correctement vos `framed` par rapport à la ligne de base, ou bien si vous avez des difficultés d'alignement / positionnement vertical, pensez à regarder l'effet de `strut`. Par exemple :

Voyez l'effet comment avec `strut=no` la première ligne à une hauteur réduire par rapport à la version `strut=yes` ce qui perturbe le positionnement par rapport à l'effet désiré.

```
\defineframed[MonCadre][width=2.25cm,align=middle]
\define[1]\DemoLoc{\rule{0pt}{3mm}%
  {\getbuffer \MonCadre[location=#1]
   {location\\ \color{darkmagenta}{\bf #1}\\location}}}
\setupbodyfont[14pt]
%\showboxes
\startbuffer
\blackrule[height=max,depth=0pt,width=3mm]%
\blackrule[height=0pt,depth=max,width=3mm]
\stopbuffer

\setupframed[MonCadre][strut=yes]
\strut
{\tt strut=yes}
\DemoLoc{bottom} \dontleavehmode \DemoLoc{top}

{\tt strut=no}
\setupframed[MonCadre][strut=no]
\DemoLoc{bottom} \dontleavehmode \DemoLoc{top}
```



12.6 Autres environnements et constructions d'intérêt

Il existe encore de nombreux environnements dans ConTeXt que je n'ai même pas mentionnés, ou seulement de manière très approximative. À titre d'exemple :

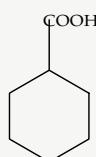
- **buffer** *Tampons (Buffers)* sont des fragments de texte stockés en mémoire pour une réutilisation ultérieure. Un *tampon* est défini quelque part dans le document avec `\startbuffer[BufferName] ... \stopbuffer` et peut être récupéré aussi souvent que souhaité à un autre endroit du document avec `\getbuffer[BufferName]`. La commande `\typebuffer[BufferName]` affiche le texte du buffer en verbatim (sans traitement par ConTeXt).

```
\startbuffer[visite]
Coucou ConTeXt.
\stopbuffer
\getbuffer[visite]
\getbuffer[visite]
```

Coucou ConTeXt. Coucou ConTeXt.

- **chemical** Cet environnement nous permet de placer des formules chimiques à l'intérieur. Si T_EX se distingue, entre autres, par sa capacité à composer correctement des textes contenant des formules mathématiques, ConTeXt a cherché dès le départ à étendre cette capacité aux formules chimiques, et dispose de cet environnement où sont activées les commandes et les structures permettant d'écrire des formules chimiques.

```
\usemodule[chemic]
\startchemical
\chemical[SIX,B,R6,RZ6][\SL{COOH}]
\stopchemical
```



- **combinations** Cet environnement nous permet de combiner plusieurs éléments flottants sur une même page. Il est particulièrement utile pour aligner différentes images externes connectées dans notre document.

```
\useMPlibrary [dum] % pour produire des images
\placefigure [here] [fig:combinations] {Un exemple de combinaison}
{\startcombination[2*2]
{\externalfigure [dummy] [height=1cm,width=4cm]} {a}
{\externalfigure [dummy] [height=1cm,width=4cm]} {b}
{\externalfigure [dummy] [height=1cm,width=4cm]} {c}
{\externalfigure [dummy] [height=1cm,width=4cm]} {d}
\stopcombination}
```



Figure 1 Un exemple de combinaison

- **formula** Il s'agit d'un environnement destiné à la composition de formules mathématiques.

```
\usemodule[newmat]

Display math:
\startformula
q = \delta \frac{\partial p}{\partial x} =
\delta(\phi) p_{vsat}(\theta) \frac{\partial}{\partial \phi} \frac{\partial \phi}{\partial x} =
\left[ \frac{\partial \phi}{\partial \theta} p_{vsat}(\theta) \right] \frac{\partial \phi}{\partial x} =
k \frac{\partial \phi}{\partial x}
\stopformula
```

Display math:

$$q = \delta \frac{\partial p}{\partial x} = \delta(\phi) p_{vsat}(\theta) \frac{\partial \phi}{\partial x} = \left[\frac{\delta_a}{\mu(\theta)} p_{vsat}(\theta) \right] \frac{\partial \phi}{\partial x} = k \frac{\partial \phi}{\partial x}$$

- **hiding** Le texte stocké dans cet environnement ne sera pas compilé et n'apparaîtra donc pas dans le document final. Ceci est utile pour désactiver temporairement la compilation de certains fragments du fichier source. On obtient la même chose en marquant une ou plusieurs lignes comme commentaire. Mais lorsque le fragment que l'on veut désactiver est relativement long, il est plus efficace que de marquer des dizaines ou des centaines de lignes du fichier source comme commentaire d'insérer la commande `\starthiding` au début du fragment, et `\stophiding` à la fin.
- **legend** Dans un contexte mathématique, TeX applique des règles différentes de sorte qu'aucun texte normal ne peut être écrit. Cependant, il arrive qu'une formule soit accompagnée d'une description des éléments qui la composent. Pour cela, il existe l'environnement `\startlegend` qui permet de placer un texte normal dans un contexte mathématique.

- **linecorrection** En général, ConTeXt gère correctement l'espace vertical entre les lignes, mais il arrive parfois qu'une ligne contienne quelque chose qui la rende incorrecte. Cela se produit principalement avec les lignes dont les fragments ont été encadrés avec `\framed`. Dans de tels cas, cet environnement ajuste l'espacement des lignes afin que le paragraphe apparaisse correctement.
- **mode** Cet environnement est destiné à inclure dans le fichier source des fragments qui ne seront compilés que si le mode approprié est actif. L'utilisation des *modes* n'est pas le sujet de cette introduction, mais c'est un outil très intéressant si l'on veut pouvoir générer plusieurs versions avec des formats différents (une version écran et une papier, ou bien une version anglaise et une française), à partir d'un seul fichier source. Voici un exemple :

```
\startmode[palatino]
  \setupbodyfont[palatino,12pt]
\stopmode

\startmode[times]
  \setupbodyfont[postscript,12pt]
\stopmode

\starttext
\input knuth
\stoptext
```

```
context --mode=palatino filename
context --mode=times   filename
```

Un environnement complémentaire à celui-ci est `\startnotmode`.

- **opposite** Cet environnement est utilisé pour composer des textes lorsque les contenus des pages de gauche et de droite sont liés.
- **quotation** Un environnement très similaire à `narrower`, destiné à insérer des citations littérales moyennement longues. L'environnement veille à ce que le texte soit cité et à ce que les marges soient augmentées pour que le paragraphe contenant la citation se détache visuellement sur la page.

Cette citation de Victor Hugo~:

```
\startquotation
Il vient une heure où protester ne
suffit plus : après la philosophie,
il faut l'action.
\stopquotation
```

Cette citation de Victor Hugo :

“Il vient une heure où protester ne suffit plus : après la philosophie, il faut l'action.”

- **standardmakeup** Cet environnement est conçu pour générer des pages de titre de chapitre, ou bien la première de couverture du document, ce qui est relativement courant dans les livres et les documents universitaires d'une certaine longueur, tels que les thèses de doctorat, les mémoires de maîtrise, etc.

Pour en savoir plus sur l'un de ces environnements (ou d'autres que je n'ai pas mentionnés), je vous suggère, en plus du [wiki](#), de suivre les étapes suivantes :

1. Cherchez des informations sur l'environnement dans le manuel de référence ConTeXt. Ce manuel ne mentionne pas tous les environnements, mais il parle de chaque élément de la liste ci-dessus.
2. Rédigez un document de test dans lequel l'environnement est utilisé.
3. Consultez la liste officielle des commandes de ConTeXt (voir [section 3.6](#)) pour connaître les options de configuration de l'environnement en question, puis testez-les pour voir exactement ce qu'elles font.

Chapitre 13

Images, tableaux et autres objets flottants

Table of Contents : [13.1 Que sont les objets flottants et que font-ils ?](#) ; [13.2 Images externes](#) ; [13.2.1 Insertion directe d'images](#) ; [13.2.2 Insertion d'une image avec \placefigure](#) ; [13.2.3 Insertion d'images intégrées dans un bloc de texte](#) ; [13.2.4 Configuration et transformation des images insérées](#) ; A Options de configuration ; B Commandes spécifiques pour transformer d'une image ; [13.3 Tableaux](#) ; [13.3.1 Idées générales sur les tableaux et leur emplacement dans le document](#) ; [13.3.2 Tableaux simples avec l'environnement \tabulate](#) ; [13.3.3 Tableaux avec l'environnement TABLE \(tableaux naturels\)](#) ; A Décrire le contenu du tableau ; B Fusionner des cellules ; C Configurer le tableau : préciser les éléments à configurer ; D Configurer le tableau : les options de configuration ; E Quelques exemples ; [13.4 Aspects communs aux images, tableaux et autres objets flottants](#) ; [13.4.1 Options d'insertion d'objets flottants](#) ; [13.4.2 Configuration des titres des objets flottants](#) ; [13.4.3 Insertion combinée de deux ou plusieurs objets](#) ; [13.4.4 Configuration générale des objets flottants](#) ; [13.5 Définition d'objets flottants supplémentaires](#) ;

Ce chapitre traite principalement des objets flottants (floats). Mais il profite de ce concept pour expliquer deux types d'objets qui ne sont pas nécessairement des objets flottants, bien qu'ils soient souvent configurés comme s'ils l'étaient : les images externes et les tableaux. À la lecture de la table des matières de ce chapitre, on pourrait penser que tout cela est très désordonné : on commence par parler des objets flottants, puis on passe aux images et aux tableaux, et on termine en parlant à nouveau des objets flottants. Les raisons de ce désordre sont d'ordre pédagogique : les images et les tableaux peuvent être expliqués sans trop insister sur le fait qu'ils sont normalement des objets flottants ; et pourtant, lorsque nous commençons à les examiner, il est très utile de découvrir que, surprise, nous connaissons déjà deux objets flottants.

13.1 Que sont les objets flottants et que font-ils ?

Si un document ne contenait que du texte *normal*, sa pagination serait relativement facile : il suffit de connaître la hauteur maximale de la zone de texte de la page pour mesurer la hauteur des différents paragraphes et savoir où insérer les sauts de page. Le problème est que dans de nombreux documents, on trouve des objets, des fragments ou des blocs indivisibles tels qu'une image, un tableau, une formule, un paragraphe encadré, etc.

Parfois, ces objets peuvent occuper une grande partie de la page, ce qui pose à son tour le problème suivant : si l'on doit l'insérer à un endroit précis du document, il se peut qu'il ne tienne pas sur la page en cours et qu'il faille l'interrompre brusquement, en laissant un grand espace vide en bas, afin que l'objet en question, et le texte qui le suit, soient déplacés sur la page suivante. Les règles d'une bonne composition indiquent cependant que, à l'exception de la dernière page d'un chapitre, il doit y avoir la même quantité de texte sur chaque page.

Il est donc conseillé d'éviter l'apparition de grands espaces verticaux vides, et les objets *flottants* sont le principal moyen d'y parvenir. Un objet flottant ne doit pas nécessairement se trouver à un endroit précis du document, mais peut *être déplacer* ou *flotter* autour de celui-ci. L'idée est de permettre à ConTeXt de décider du meilleur endroit, du point de vue de la pagination, pour placer de tels objets, voire de les autoriser à se déplacer vers une autre page ; mais en essayant toujours de ne pas trop s'éloigner du point d'inclusion dans le fichier source.

Par conséquent, il n'y a pas d'objet qui *doit* être un flottant (en soi). Mais il y a des objets qui devront occasionnellement être des flottants. Dans tous les cas, la décision appartient à l'auteur ou à la personne chargée de la composition, s'il s'agit de deux personnes différentes.

Il ne fait aucun doute que le fait de permettre le changement de l'emplacement exact d'un objet indivisible facilite grandement la composition de pages joliment équilibrées ; mais le problème qui en découle est que, puisque nous ne savons pas exactement où cet objet se retrouvera au moment où nous écrivons le fichier source, il est difficile d'y faire référence. Ainsi, par exemple, si je viens de mettre dans mon document une commande qui insère une image et que dans le paragraphe suivant je veux la décrire et écrire quelque chose à son sujet comme : « Comme vous pouvez le voir sur la figure précédente », lorsque la figure *flotte*, elle pourrait bien être placée *après* ce que je viens d'écrire et le résultat aboutir à une incohérence : le lecteur cherche une image *avant* le texte qui y fait référence et ne la trouve pas car après avoir flotté, l'image s'est retrouvée après cette référence.

Ce problème est résolu par la *numérotation* des objets flottants (après les avoir répartis en catégories), de sorte qu'au lieu de se référer à une image comme « l'image précédente » ou « l'image suivante », nous nous y référerons comme « image 1.3 », puisque nous pouvons utiliser le mécanisme de référence interne de ConTeXtpour

nous assurer que le numéro de l'image est toujours maintenu à jour (voir section ??). La numérotation de ces types d'objets, d'autre part, permet de créer assez facilement un index de ceux-ci (index des tableaux, graphiques, images, équations, etc.). Pour savoir comment faire, voir ([section 8.2](#)).

Le mécanisme de traitement des objets flottants dans ConTeXt est assez sophistiqué et parfois si abstrait qu'il peut ne pas convenir aux débutants. C'est pourquoi, dans ce chapitre, je commencerai par l'expliquer en utilisant deux cas particuliers : les images et les tableaux. Ensuite, j'essaierai de généraliser quelque peu pour que nous puissions comprendre comment étendre le mécanisme à d'autres types d'objets.

13.2 Images externes

²⁵ Les modules d'extension de ConTeXt donnent des fonctionnalités supplémentaires mais ne sont pas inclus dans cette introduction.

²⁶ Il s'agit d'un langage de programmation graphique destiné à fonctionner avec les systèmes basés sur TeX. Il s'agit d'un « acronyme récurrent » de la phrase allemande « TiKZ ist keinen Zeichenprogramm » qui se traduit par : « TiKZ n'est pas un programme de dessin ». Les acronymes récursifs sont une sorte de blague de programmeurs. En laissant de côté MetaPost (que je ne sais pas utiliser), je pense que TiKZ est un excellent système pour programmer des images

Comme le lecteur le sait à ce stade (puisque cela a été expliqué dans section 1.5), MetaPost est parfaitement intégré à ConTeXt (sous le nom de Metafun) et peut générer des images et des graphiques qui sont programmés de la même manière que les transformations de texte sont programmées. Il existe également un module d'extension pour ConTeXt²⁵ qui lui permet de fonctionner avec TiKZ.²⁶ Mais ces images ne sont pas traitées dans cette introduction (car cela obligerait probablement à doubler sa longueur). Je fais ici référence à l'utilisation d'images externes, qui résident dans un fichier sur notre disque dur ou qui sont téléchargées directement de l'Internet par ConTeXt.

13.2.1 Insertion directe d'images

Pour insérer directement une image (pas comme un objet flottant), nous utilisons la commande `\externalfigure`

```
\externalfigure [^1] [^2] [...] ^3 [...] ^4  
1 FILE ^5  
2 NAME ^6  
3 inherits: \setupexternalfigure
```

- 1 et 2 peuvent soit le nom du fichier contenant l'image, soit l'adresse Web d'une image sur Internet, soit un nom symbolique que nous avons précédemment associé à une image à l'aide de la commande `\useexternalfigure` dont le format est similaire à celui de `\externalfigure` bien qu'elle prenne un premier argument avec le nom symbolique qui sera associé à l'image en question.
- 3 est un argument facultatif qui nous permet d'appliquer certaines transformations à l'image avant qu'elle ne soit insérée dans notre document. Nous examinerons cet argument de plus près dans la section 13.2.4.

Les formats d'image autorisés sont pdf, mps, jpg, png, jp2, jbig, jbig2, jb2, svg, eps, gif, tif. ConTeXt peut gérer directement huit d'entre eux, tandis que les autres (svg, eps, gif, tif) doivent être convertis avec un outil externe avant de les ouvrir, qui change en fonction du format et doit donc être installé sur le système pour que ConTeXt puisse manipuler ce type de fichiers.

Parmi les formats pris en charge par `\externalfigure` figurent également certains formats vidéo. En particulier : QuickTime (extension .mov), Flash Video (extension .flv) et MPeg 4 (extension .mp4). Mais la plupart des lecteurs de PDF ne savent pas comment traiter les fichiers PDF contenant de la vidéo. Je ne peux pas me prononcer sur ce point, car je n'ai pas fait de tests.

Il n'est pas nécessaire d'indiquer l'extension du fichier : ConTeXt recherchera un fichier avec le nom spécifié et l'une des extensions des formats d'image connus. S'il y a plusieurs candidats, c'est d'abord le format PDF qui est utilisé s'il existe, et en son absence le format MPS (graphiques générés par MetaPost). En l'absence de ces deux formats, l'ordre suivant est suivi : jpeg, png, jpeg 2000, jbig et jbig2.

Si le format réel de l'image ne correspond pas à l'extension du fichier qui la stocke, ConTeXt ne peut pas l'ouvrir à moins que nous n'indiquions le format réel de l'image à l'aide de l'option `method`.

Si l'image n'est pas placée seule en dehors d'un paragraphe, mais qu'elle est intégrée à un paragraphe de texte, et que sa hauteur est supérieure à l'interligne, la ligne sera ajustée pour éviter que l'image ne chevauche les lignes précédentes :

Si l'image est intégrée à un paragraphe de texte, la hauteur de ligne sera ajustée pour éviter que l'image `\externalfigure [cow-brown] [width=3em]` ne chevauche les lignes précédentes et que l'ensemble reste lisible.

Si l'image est intégrée à un paragraphe de texte, la hauteur de ligne sera ajustée pour éviter que l'image  ne chevauche les lignes précédentes et que l'ensemble reste lisible.

Par défaut, ConTeXt cherche les images dans le répertoire de travail, dans son répertoire parent et dans le répertoire parent de ce répertoire. Nous pouvons indiquer l'emplacement d'un répertoire contenant les images avec lesquelles nous allons travailler en utilisant l'option `directory` de la commande `\setupexternalfigures`, qui ajoutera ce répertoire au chemin de recherche. Si nous voulons que la recherche soit effectuée uniquement dans le répertoire d'images, nous devons également définir l'option `location`. Ainsi, par exemple, pour que notre document recherche toutes les images dont nous avons besoin dans le répertoire « `img` », nous devons écrire :

Par défaut, ConTeXt cherche les images dans le répertoire de travail, dans son répertoire parent et dans le répertoire parent de ce dernier. Nous pouvons indiquer l'emplacement d'un répertoire contenant les images avec lesquelles nous allons travailler en utilisant l'option `directory` de la commande `\setupexternalfigures`, qui ajoutera ce répertoire au chemin de recherche. Si nous voulons que la recherche soit effectuée uniquement dans le répertoire d'images, nous devons également définir l'option `location`. Ainsi, par exemple, pour que notre document recherche toutes les images dont nous avons besoin dans le répertoire « `/Images` », nous devons écrire :

```
\setupexternalfigures [directory=~/Image]
```

Dans l'option `directory` de `\setupexternalfigures`, nous pouvons inclure plus d'un répertoire, en les séparant par des virgules ; mais dans ce cas, nous devons mettre les répertoires entre accolades. Par exemple : « `directory={img, ~/images}` ».

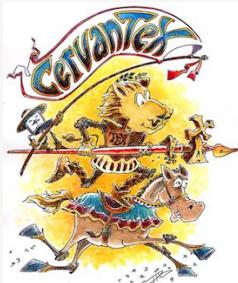
Dans `directory`, nous utilisons toujours le caractère « `/` » comme séparateur entre les répertoires ; y compris dans Microsoft Windows dont le système d'exploitation utilise le caractère « `\` » comme séparateur de répertoires.

`\externalfigure` est également capable d'utiliser des images hébergées sur Internet. Ainsi, par exemple, l'extrait suivant insère le logo CervanTeX directement depuis Internet dans le document. Il s'agit du groupe d'utilisateurs hispanophones de TeX :

²⁷ Cette dernière est ma conclusion, étant donné que parmi les options de placement, il y a celles comme force ou split qui vont à l'encontre de la véritable notion d'objet flottant.

\externalfigure

[<http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg>]



Lorsqu'un document contenant un fichier distant est compilé pour la première fois, il est téléchargé du serveur et stocké dans le répertoire de cache de LuaTeX. Ce fichier en cache est utilisé lors des compilations suivantes. Normalement, l'image distante est téléchargée à nouveau si l'image dans le cache est plus ancienne que 1 jour. Pour modifier ce seuil, consultez le [wiki ConTeXt](#).

Si ConTeXt ne trouve pas l'image qui devrait être insérée, aucune erreur n'est générée, mais à la place de l'image, un bloc de texte sera inséré avec des informations sur l'image qui devrait s'y trouver. La taille de ce bloc sera celle de l'image (si elle est connue de ConTeXt) ou, sinon, une taille standard.

```
\externalfigure  
[image-perdueA.jpg]  
\externalfigure  
[image-perdueB.jpg]  
[height=2cm]
```

```
name: image-perdueA.jpg  
file: image-perdueA.jpg  
state: unknown
```

```
name: image-perdueB.jpg  
file: image-perdueB.jpg  
state: unknown
```

13.2.2 Insertion d'une image avec \placefigure

Les images peuvent être insérées directement. Mais il est préférable de le faire avec [`\placefigure`](#). Cette commande à plusieurs implications vis à vis de ConTeXt. Elle permet :

- d'indiquer que l'on insère une image qui doit être incorporée à la liste des images du document qui pourra ensuite être utilisée, si on le souhaite, pour produire un index des images.
- d'attribuer un numéro à l'image, facilitant ainsi les références internes à celle-ci.
- d'ajouter un titre à l'image, en créant un bloc combinant l'image et son titre, ce qui les rend indissociables.
- de définir automatiquement l'espace blanc (horizontal et vertical) entourant l'image et nécessaire sa bonne visualisation.
- de positionner l'image à l'endroit indiqué, en faisant *circuler* le texte autour d'elle si nécessaire.
- pour convertir l'image en objet flottant si cela est possible, en tenant compte de ses spécifications de taille et d'emplacement.²⁷

La syntaxe de cette commande est la suivante :

```
\placefigure[Options] [Etiquette] {Titre} {CommandeInsererImage}
```

Elle hérite de la commande `\placefloat`.

```
\placefloat [...] [...] [...] {...} {...}
          OPT      OPT
1 SINGULAR
2 split always left right inner outer backspace cutspace inleft inright
  inmargin leftmargin rightmargin lefthead rightedge innermargin outermargin
  inneredge outeredge text opposite reset height depth [-+]line halffline
  grid high low fit 90 180 270 nonumber none local here force margin
  [-+]hang hanging tall both middle offset top bottom auto page leftpage
  rightpage somewhere effective header footer tblr lrtb tbrl rltb fxtb btlr
  lrtb btlr rltb fxbt fixd
3 REFERENCE
4 TEXT
5 CONTENT
```

The various arguments have the following meanings :

- *Options* sont un ensemble d'indications qui font généralement référence à l'endroit où placer l'image. Comme ces options sont les mêmes dans cette commande et dans d'autres, je les expliquerai ensemble plus tard (dans section 13.4.1). Pour l'instant, je vais utiliser l'option `here` à titre d'exemple. Elle indique à ConTeXt que, dans la mesure du possible, il doit placer l'image exactement à l'endroit du document où se trouve la commande qui l'insère.
- *Etiquette* est une chaîne de texte permettant de se référer en interne à cet objet afin de pouvoir y faire référence (voir section 9.2).
- *Titre* est le texte du titre à ajouter à l'image.
- *CommandeInsererImage* est la commande qui insère l'image (on peut aussi juste mettre... du texte.)

Par exemple :

```
\setupfloat[figure][location=center]
Imaginons un texte introductif.
\placefigure
  [here]
  [fig:cm2016group]
  {\ConTeXt\ Meeting Group 2016}
  {\externalfigure[https://meeting.contextgarden.net/2016/img/cm2016-group-photo.jpg]}
Avez vous vu la \in{figure}[fig:cm2016group] ?
\blank[big]
{\bfa Liste des figures}
\blank[small]
\placelistoffigures
```

Imaginons un texte introductif.

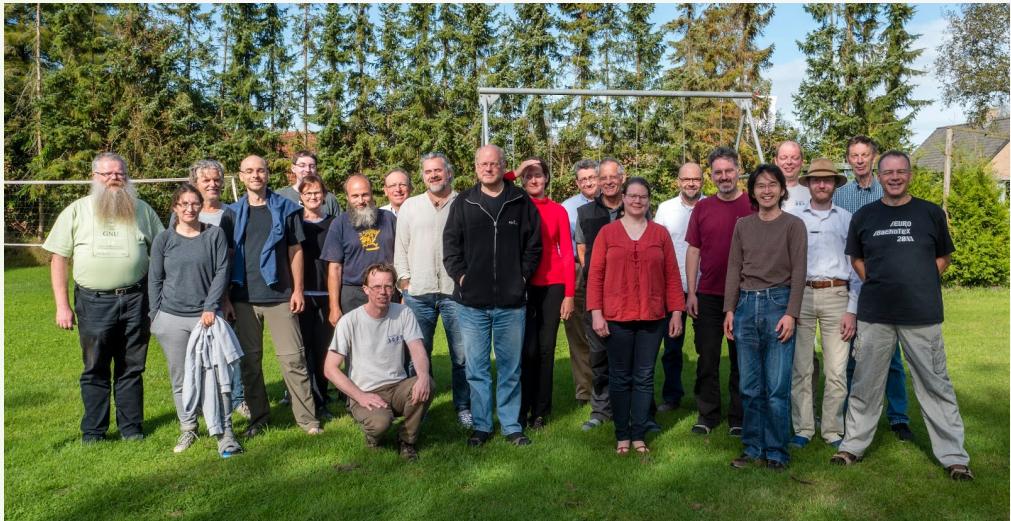


Figure 1 ConTeXt Meeting Group 2016

Avez vous vu la figure 1 ?

Liste des figures

1 ConTeXt Meeting Group 2016

0

Comme nous pouvons le voir dans l'exemple, en insérant l'image (ce qui, soit dit en passant, a été fait directement à partir d'une image hébergée sur Internet), il y a quelques changements par rapport à ce qui se passe en utilisant directement la commande `\externalfigure`. Un espace vertical est ajouté, l'image est centrée et un titre est ajouté. Il s'agit de modifications *externes* évidentes à première vue. D'un point de vue interne, la commande a également produit d'autres effets non moins importants :

- Tout d'abord, l'image a été insérée dans la « liste d'images » que ConTeXt maintient en interne pour les objets insérés dans le document. Cela signifie donc que l'image apparaîtra dans l'index des images qui peut être généré avec `\place-list[figure]` (voir section 8.2), bien qu'il existe deux commandes spécifiques pour générer l'index d'images, à savoir `\placelistoffigures` ou `\completelistoffigures`.
- Ensuite, l'image a été liée à l'étiquette qui a été ajoutée comme deuxième argument de la commande `\placefigure`, ce qui signifie qu'à partir de maintenant, nous pouvons faire des références internes à cette image en utilisant cette étiquette (voir section ??).
- Enfin, l'image est devenue un flottant, ce qui signifie que si, pour des besoins de composition (pagination), elle devait être déplacée, ConTeXt modifierait son emplacement.

En fait, `\placefigure`, malgré son nom, n'est pas seulement utilisé pour insérer des images. Nous pouvons insérer n'importe quoi avec elle, y compris du texte. Cependant, le texte ou les autres éléments insérés dans le document avec `\placefigure`, seront traités *comme s'il s'agissait d'une image*, même s'ils ne le sont pas ; ils seront ajoutés à la liste des images gérée en interne par ConTeXt, et nous pouvons appliquer des transformations similaires à celles que nous utilisons pour les images telles que la mise à l'échelle ou la rotation, le cadrage, etc. Ainsi, l'exemple suivant :

```
\setupfloat [figure] [location=center]
\placefigure
  [here, force]
  [fig:testtext]
  {\tex{\placefigure} avec des transformations textuelles.}
  {\rotate[rotation=35]{\color{darkred}{\bf Excellent !!!}}}
```

Figure 1 `\placefigure` avec des transformations textuelles.

13.2.3 Insertion d'images intégrées dans un bloc de texte

À l'exception des très petites images, qui peuvent être intégrées dans une ligne sans trop perturber l'espacement des paragraphes, les images sont généralement insérées dans un paragraphe qui ne contient qu'elles (ou dit autrement, l'image peut être considérée comme un paragraphe à part entière). Si l'image est insérée avec `\placefigure` et que sa taille le permet, en fonction de ce que nous avons indiqué concernant son placement (voir section 13.4.1), ConTeXt permettra au texte du

paragraphe précédent et des paragraphes suivants de circuler autour de l'image. Cependant, si nous voulons nous assurer qu'une certaine image ne sera pas séparée d'un certain texte, nous pouvons utiliser l'environnement `figuretext` dont la syntaxe est la suivante :

```
Texte
\setupfloat [figure] [location=center]
\startfiguretext
[left]
[fig:testtext]
{\tex{placefigure} avec des transformations textuelles.}
{\rotate[rotation=20]{\color{darkred}{\bf Excellent ce texte image!!!}}}
Texte d'accompagnement qui peut contenir beaucoup d'information selon
l'inspiration de l'auteur.
\stopfiguretext
```

Texte

Texte d'accompagnement qui peut contenir beaucoup d'information selon l'inspiration de l'auteur.

Excellent ce texte image!!!

Figure 1 `\placefigure` avec des transformations textuelles.

Les arguments de l'environnement sont exactement les mêmes que pour `\placefigure` et ont la même signification. Mais ici, les options ne sont plus des options de placement d'un objet flottant, mais des indications concernant l'intégration de l'image dans le paragraphe ; ainsi, par exemple, « `left` » signifie ici que l'image sera placée sur la gauche tandis que le texte s'écoulera vers la droite, tandis que « `left, bottom` » signifiera que l'image doit être placée sur le côté inférieur gauche du texte qui lui est associé.

Le texte écrit dans l'environnement est ce qui circulera autour de l'image.

```

Texte
\setupfloat [figure] [location=center]
\startfiguretext
  [right,low] % or middle
  [fig:testtext]
  {\tex{placefigure} avec des transformations textuelles.}
  {\rotate[rotation=20]{\color{darkred}{\bf Excellent ce texte image!!!}}}
Texte d'accompagnement qui peut contenir beaucoup d'information selon
l'inspiration de l'auteur.
\stopfiguretext

```

Texte

Excellent ce texte image!!!

Texte d'accompagnement qui peut contenir beaucoup d'information selon l'inspiration de l'auteur.

Figure 1 \placefigure avec des transformations textuelles.

13.2.4 Configuration et transformation des images insérées

A. Options de configuration

Le dernier argument de la commande `\externalfigure` nous permet d'indiquer certains ajustements sur l'image insérée. Nous pouvons effectuer ces ajustements :

- de façon générale (pour toutes les images insérées dans le document) ou pour toutes les images insérées à partir d'un certain point. Dans ce cas, nous effectuons l'ajustement avec la commande `\setupexternalfigures`.
- Pour une image spécifique que l'on souhaite insérer plusieurs fois dans le document. Dans ce cas, l'ajustement se fait dans le dernier argument de la commande `\useexternalfigure` qui associe une figure externe à un nom symbolique.
- Au moment précis où nous insérons une image spécifique. Dans ce cas, l'ajustement est effectué dans la commande `\externalfigure` elle-même.

Les modifications de l'image qui peuvent être obtenues par cette voie sont les suivantes :

Modification de la taille de l'image. Nous pouvons le faire :

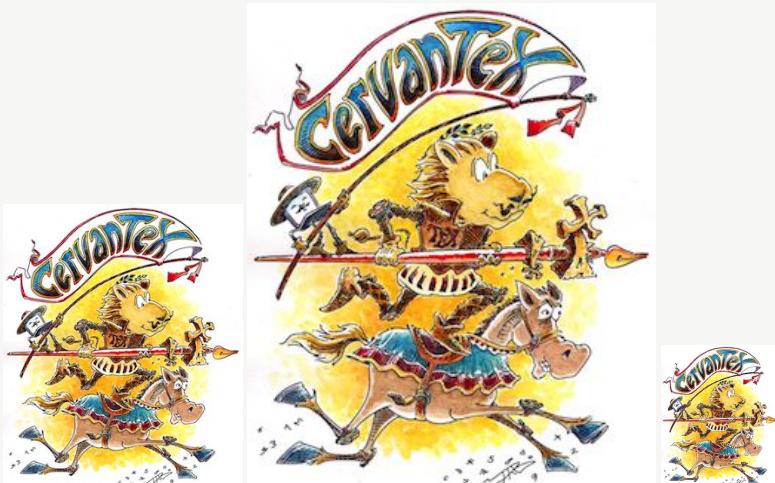
- En attribuant une largeur ou une hauteur précise, ce qui se fait respectivement avec les options `width` et `height` ; si une seule des deux valeurs est ajustée, l'autre est automatiquement adaptée pour maintenir les proportions initiales de l'image.

On peut attribuer une hauteur ou une largeur précise, ou l'indiquer en pourcentage de la hauteur de la page ou de la largeur de la ligne. Par exemple pour faire en sorte que l'image ait une largeur égale à 40% de la largeur de ligne on indiquera `width=.4\textwidth`

- *Mise à l'échelle de l'image*: L'option `xscale` permet de mettre l'image à l'échelle horizontalement ; `yscale` permet de le faire verticalement et `scale` permet de le faire horizontalement et verticalement. Ces trois options attendent un nombre représentatif du facteur d'échelle multiplié par 1000. En d'autres termes : `scale=1000` laissera l'image dans sa taille d'origine, tandis que `scale=500` la réduira de moitié, et `scale=2000` la doublera.

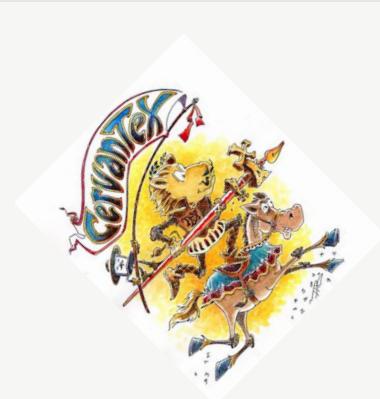
Une mise à l'échelle conditionnelle, qui n'est appliquée que si l'image dépasse certaines dimensions, est obtenue avec les options `maxwidth` et `maxheight` qui prennent une dimension. Par exemple, `maxwidth=.2\textwidth` ne redimensionnera l'image que si elle dépasse 20% de la largeur de la ligne.

```
\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg]
\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg]
[width=0.4\textwidth]
\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg]
[scale=500]
```



Tourner l'image. Pour faire tourner l'image, nous utilisons l'option `orientation` qui prend un nombre représentatif du nombre de degrés de rotation qui sera appliqué. La rotation se fait dans le sens inverse des aiguilles d'une montre.

```
\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg]
[orientation=45]
\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg]
[orientation=-10]
```



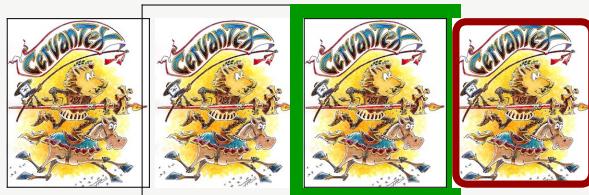
Le wiki implique que les rotations qui peuvent être obtenues avec cette option doivent être des multiples de 90 (90, 180 ou 270) mais pour obtenir une rotation différente, il faudrait utiliser la commande `\rotate`. Cependant, je n'ai eu aucun problème à appliquer une rotation de 45 degrés à une image avec seulement `orientation=45`, sans avoir besoin d'utiliser la commande `\rotate`.

Encadrer l'image. On peut également entourer l'image d'un cadre à l'aide de l'option `frame=on`, et configurer sa couleur (`framecolor`), la distance entre le cadre et l'image (`frameoffset`), l'épaisseur du trait qui dessine le cadre (`rulethickness`) ou la forme de ses coins (`framecorner`) qui peuvent être arrondis (`round`) ou rectangulaires.

```

\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg]
[scale=600,frame=on]
\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg]
[scale=600,frame=on,frameoffset=2mm]
\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg]
[scale=600,frame=on,backgroundoffset=2mm,background=color,backgroundcolor=darkgreen]
\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg]
[scale=600,frame=on,framecolor=darkred,rulethickness=3pt,framecorner=round]

```



Autres aspects configurables des images. Outre les aspects déjà vus, qui impliquent une transformation de l'image à insérer, il est possible, à l'aide de `\setupexternalfigures`, de configurer d'autres aspects, tels que l'endroit où chercher l'image (option `directory`), si l'image doit être recherchée uniquement dans le répertoire indiqué (`location=global`) ou si elle doit également inclure le répertoire de travail et ses répertoires parents (`location=local`), si l'image sera ou non interactive (`interaction`), etc.

B. Commandes spécifiques pour transformer d'une image

Il existe trois commandes dans ConTeXt qui produisent une certaine transformation dans une image et peuvent être utilisées en combinaison avec `\externalfigure`. Il s'agit de :

- *Image miroir* : obtenue avec la commande `\mirror`.
- *Découpe* : cette opération est réalisée à l'aide de la commande `\clip` lorsque les dimensions de la largeur (`width`), de la hauteur (`height`), du décalage horizontal (`hoffset`) et du décalage vertical (`voffset`) sont données. Par exemple :
- *Rotation* : Une troisième commande capable d'appliquer des transformations à une image est la commande `\rotate`. Elle peut être utilisée en conjonction avec `\externalfigure` mais normalement, cela ne serait pas nécessaire étant donné que cette dernière dispose, comme nous l'avons vu, de l'option `orientation` qui produit le même résultat.

```
\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg] [scale=750]
\mirror{\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg] [scale=750]}
\clip [width=2cm, height=2cm] {\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg] [scale=750]}
\clip [width=2cm, height=2cm,hoffset=1cm,voffset=1cm] {\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg] [scale=750]}
\rotate[rotation=20]{\externalfigure
[http://www.cervantex.es/files/cervantex/cervanTeXcolor-small.jpg] [scale=750]}
\rotate[rotation=45]{\clip [width=3cm, height=1cm,hoffset=1.5cm]{\bf Coucou \ConTeXt}}
```



ou ConTeXt

L'utilisation typique de ces commandes concerne les images, mais elles agissent en fait sur les *boîtes* (*boxes*). C'est pourquoi nous pouvons les appliquer à n'importe quel texte en l'enfermant simplement dans une boîte (ce que la commande fait automatiquement) voyez le dernier exemple.

13.3 Tableaux

13.3.1 Idées générales sur les tableaux et leur emplacement dans le document

Les tableaux sont des objets structurés qui contiennent du texte, des formules ou même des images disposés dans une série de *cellules* qui permettent de visualiser graphiquement la relation entre le contenu de chaque cellule. Pour ce faire, les informations sont organisées en lignes et en colonnes : toutes les données (ou entrées) d'une même ligne ont une certaine relation entre elles, de même que toutes les données (ou entrées) d'une même colonne. Une cellule est l'intersection d'une ligne et d'une colonne, comme le montre la [figure 13.1](#).

Les tableaux sont idéaux pour afficher des textes ou des données qui sont liés les uns aux autres, car comme chacun est enfermé dans sa propre cellule, même si son contenu ou celui des autres cellules change, la position relative de l'un par rapport aux autres ne changera pas.

De toutes les tâches liées à la composition d'un texte, la création de tableaux est la seule qui, à mon avis, est plus facile à réaliser dans un programme graphique (type traitement de texte) que dans ConTeXt. Parce qu'il est plus facile de *dessiner* le tableau (ce que l'on fait dans un programme de traitement de texte) que de le *décrire* (ce que l'on fait quand on travaille avec ConTeXt). Chaque changement de ligne et de colonne nécessite la présence d'une commande, ce qui signifie que l'implémentation du tableau prend beaucoup plus de temps, au lieu de simplement dire combien de lignes et de colonnes nous voulons.

ConTeXt a trois mécanismes différents pour produire des tableaux ; l'environnement `tabulate` qui est recommandé pour les tableaux simples et qui est le plus directement inspiré des tableaux TeX ; les tableaux dits *naturels* (*natural tables*), inspirés des tableaux HTML, adaptés aux tableaux ayant des besoins de conception particuliers où, par exemple, toutes les lignes n'ont pas le même nombre de colonnes ; et les tableaux dits *extrêmes* (*extreme tables*), clairement basés sur XML et recommandés pour les tableaux moyens ou longs qui prennent plus d'une page. Des trois, je n'expliquerai que les deux premiers. Les tableaux naturels sont également raisonnablement bien expliqués dans la « ConTeXt Mark IV an excursion », et pour les *extreme tables* il y a une [documentation officielle](#) à leur sujet dans la documentation de la distribution « ConTeXt Standalone ».

Il se passe quelque chose de similaire à ce qui se passe avec les images pour les tableaux : il suffit d'écrire les commandes nécessaires à un moment donné du document pour générer un tableau et celui-ci sera inséré à cet endroit précis, ou bien nous pouvons utiliser la commande `\placetable` pour insérer un tableau. Cette méthode présente quelques avantages :

	Colonne 1	Colonne 2	Colonne 3
Ligne 1			
Ligne 2			
Ligne 3			
Ligne 4			

Figure 13.1 Exemple de table simple

- ConTEXt numérote le tableau et l'ajoute à la liste des tableaux permettant des références internes au tableau (par sa numérotation), l'incluant dans un éventuel index des tableaux.
- Nous gagnerons en flexibilité dans le placement des tableaux dans le document, facilitant ainsi la tâche de la pagination.

Le format de `\placetable` est similaire à ce que nous avons vu pour `\placefigure` (voir section 13.2.2) :

```
\placetable[Options] [Etiquette] {Titre} {CommandeInsererImage}
```

Je renvoie aux sections 13.4.1 et 13.4.2 concernant les options relatives au placement des tables et à la configuration du titre. Parmi les options, il y en a une, cependant, qui semble être conçue exclusivement pour les tableaux. Il s'agit de l'option « `split` » qui, lorsqu'elle est définie, autorise ConTEXt à étendre le tableau sur deux pages ou plus, auquel cas le tableau ne peut plus être un objet flottant simple.

De manière générale, nous pouvons définir la configuration des tableaux avec la commande `\setuptables`. De plus, comme pour les images, il est possible de générer un index des tables avec la commande `\placelistoftables` ou `\completelistoftables`. Voir à ce sujet section 8.2.2.

13.3.2 Tableaux simples avec l'environnement `\tabulate`

Les tableaux les plus simples sont ceux réalisés avec l'environnement *tabulate* dont le format est :

```
\starttabulate[Configuration de la disposition des colonnes du tableau]
... % Table contents
...
\stoptabulate
```

Où l'argument pris entre crochets décrit (en code) le nombre de colonnes que le tableau aura, et indique (parfois indirectement) leur largeur. Je dis que l'argument décrit le dessin *en code*, parce qu'à première vue il semble très cryptique : il consiste en une séquence de caractères, chacun ayant une signification particulière. Je vais l'expliquer petit à petit et par étapes, car je pense que de cette façon il est plus facile à comprendre.

C'est le cas typique dans lequel le nombre énorme d'aspects que nous pouvons configurer signifie que nous avons besoin de beaucoup de texte pour le décrire. Cela semble être diablement difficile. En fait, pour la plupart des tableaux construits dans la pratique, les points 1 et 2 sont suffisants. Les autres sont des possibilités supplémentaires dont il est utile de connaître l'existence, mais qu'il n'est pas indispensable de connaître pour composer un tableau.

1. **Délimiteur de colonnes** : le caractère « | » est utilisé pour délimiter les colonnes de la table. Ainsi, par exemple, « [|1T|rB|] » décrira un tableau avec deux colonnes, dont l'une aura les caractéristiques associées aux indicateurs « 1 » et « T » (que nous verrons immédiatement après) et la deuxième colonne aura les caractéristiques associées aux indicateurs « r » et « B ». Un tableau simple à trois colonnes alignées à gauche, par exemple, serait décrit comme suit : « [|1|1|1|] ».
2. **Détermination de la nature fondamentale des cellules d'une colonne** : La première chose à déterminer lorsque nous construisons notre tableau est de savoir si nous voulons que le contenu de chaque cellule soit écrit sur une seule ligne, ou si, au contraire, si le texte d'une colonne est trop long, nous voulons que notre tableau le répartisse sur deux lignes ou plus. Dans l'environnement `tabulate`, cette question n'est pas tranchée cellule par cellule mais est considérée comme une caractéristique des colonnes.
 - a. *Cellules n'occupant qu'une ligne* : Si le contenu des cellules d'une colonne, quelle que soit leur longueur, doit être écrit sur une seule ligne, nous devons spécifier l'alignement du texte dans la colonne, qui peut être à gauche (« 1 », de *left*), à droite (« r », de *right*) ou centré (« c », de *center*).

En principe, ces colonnes seront aussi larges que nécessaire pour s'adapter à la cellule la plus large. Mais nous pouvons limiter la largeur de la colonne avec le spécificateur « `w(Width)` ». Par exemple, « [|rw(2cm)|c|c|] » décrira un tableau avec deux colonnes, la première alignée à droite et d'une largeur exacte de 2 centimètres, et les deux autres centrées et sans limitation de largeur.

Il convient de noter que la limitation de la largeur des colonnes à une ligne peut entraîner le chevauchement du texte d'une colonne avec celui de la colonne suivante. Je vous conseille donc, lorsque vous avez besoin de colonnes de largeur fixe, de toujours utiliser des colonnes de cellules multilignes.
 - b. *Cellules pouvant occuper plus d'une ligne si nécessaire* : le spécificateur « `p` » génère des colonnes dans lesquelles le texte de chaque cellule occupera autant de lignes que nécessaire. Si l'on indique simplement « `p` », la largeur de la colonne sera la pleine largeur disponible. Mais il est également possible d'indiquer « `p(Width)` », auquel cas la largeur sera celle expressément spécifiée. Ainsi, les exemples suivants :

```
\startbuffer
\NC Colonne 1\NC Colonne 2\NC Colonne 3\NC\NR
\NC Ligne 1 \NC Texte 1.2\NC Texte 1.3\NC\NR
\NC Ligne 2 \NC Texte 2.2\NC Texte 2.3\NC\NR
\NC Ligne 3 \NC Texte 3.2\NC Texte 3.3\NC\NR
\NC Ligne 4 \NC Texte 4.2\NC Texte 4.3\NC\NR
\stopbuffer

\starttabulate[|l|r|p|]
\getbuffer
\stoptabulate
\blank[big]
\starttabulate[|r|p(4cm)|cw(.2\textwidth)|]
\getbuffer
\stoptabulate
\blank[big]
\starttabulate[|p|p|p|]
\getbuffer
\stoptabulate
```

Colonne 1	Colonne 2	Colonne 3
Ligne 1	Texte 1.2	Texte 1.3
Ligne 2	Texte 2.2	Texte 2.3
Ligne 3	Texte 3.2	Texte 3.3
Ligne 4	Texte 4.2	Texte 4.3

Colonne 1	Colonne 2	Colonne
Ligne 1	Texte 1.2	Texte 1.3
Ligne 2	Texte 2.2	Texte 2.3
Ligne 3	Texte 3.2	Texte 3.3
Ligne 4	Texte 4.2	Texte 4.3

Colonne 1	Colonne 2	Colonne 3
Ligne 1	Texte 1.2	Texte 1.3
Ligne 2	Texte 2.2	Texte 2.3
Ligne 3	Texte 3.2	Texte 3.3
Ligne 4	Texte 4.2	Texte 4.3

Le premier exemple créera un tableau avec trois colonnes, la première et la deuxième d'une seule ligne, alignées, respectivement, à gauche et à droite, et la troisième, qui occupera la largeur restante et la hauteur nécessaire pour accueillir tout son contenu. Dans le deuxième exemple, la deuxième colonne mesurera exactement quatre centimètres de large, quel que soit son contenu (si elle ne tient pas dans cet espace, elle occupera plus d'une ligne), et la troisième a une largeur proportionnelle à la largeur maximale de la ligne. Dans le dernier exemple, il y aura trois colonnes de largeur égales occupant l'espace disponible.

Notez qu'en réalité, si une cellule est un quadrilatère, ce que fait le spécificateur « p » est d'autoriser une hauteur variable pour les cellules d'une colonne, en fonction de la longueur du texte.

3. **Ajout d'indications à la description de la colonne, sur le style et la variante de police à utiliser** : une fois que la nature de base de la colonne (largeur et hauteur, automatique ou fixe, des cellules) a été décidée, on peut encore ajouter, dans la description du contenu de la colonne, un caractère représentatif du *formatage* dans lequel il doit être écrit. Ces caractères peuvent être : « B » pour le gras, « I » pour l'italique, « S » pour l'oblique, « R » pour la lettre de style romain ou « T » pour la lettre de style machine à écrire.

```
\startbuffer
\NC Colonne 1\NC Colonne 2\NC Colonne 3\NC\NR
\NC Ligne 1  \NC Texte 1.2\NC Texte 1.3\NC\NR
\NC Ligne 2  \NC Texte 2.2\NC Texte 2.3\NC\NR
\NC Ligne 3  \NC Texte 3.2\NC Texte 3.3\NC\NR
\NC Ligne 4  \NC Texte 4.2\NC Texte 4.3\NC\NR
\stopbuffer

\starttabulate[|1B|rI|pT|]
\getbuffer
\stoptabulate
```

Colonne 1	Colonne 2	Colonne 3
Ligne 1	Texte 1.2	Texte 1.3
Ligne 2	Texte 2.2	Texte 2.3
Ligne 3	Texte 3.2	Texte 3.3
Ligne 4	Texte 4.2	Texte 4.3

4. **Autres aspects supplémentaires qui peuvent être spécifiés dans la description des colonnes du tableau :**

- *Colonnes avec formules mathématiques* : les spécificateurs « m » et « M » permettent d'activer le mode mathématique dans une colonne sans avoir à le spécifier dans chacune de ses cellules. Les cellules de cette colonne ne pourront pas contenir de texte normal.

```
\startbuffer
\NC Colonne 1\NC Colonne 2\NC Colonne 3\NC\NR
\NC Ligne 1  \NC \int_{a}^{b} f(x) \NC\int_a^b f(x)\NC\NR
\NC Ligne 2  \NC c^2 = a^2 + b^2 \NC c^2 = a^2 + b^2 \NC\NR
\stopbuffer

\starttabulate[|1B|m|M|]
\getbuffer
\stoptabulate
```

Colonne 1	Colonne2	Colonne3
Ligne 1	$\int_a^b f(x)$	$\int_a^b f(x)$
Ligne 2	$c^2 = a^2 + b^2$	$c^2 = a^2 + b^2$

Bien que T_EX, le précurseur de ConT_EXt ait été créé pour la composition de tout type de mathématiques, je n'ai pratiquement rien dit jusqu'à présent sur l'écriture des mathématiques. Dans le mode mathématique (que je n'expliquerai pas), ConT_EXt modifie nos règles normales et utilise même des polices différentes. Le mode maths a deux variations : l'une que nous pourrions appeler *linéaire* dans la mesure où la formule est logée dans une ligne contenant du texte normal (indicateur « `m` »), et le *mode maths complet* qui affiche les formules dans un environnement où il n'y a pas de texte normal. La principale différence entre les deux modes, dans un tableau, est essentiellement la taille dans laquelle la formule sera écrite et l'espace horizontal et vertical qui l'entoure.

- Ajouter un espace blanc supplémentaire autour du contenu des cellules d'une colonne : avec les indicateurs « `in` », « `jn` » et « `kn` », nous pouvons ajouter un espace blanc supplémentaire à gauche du contenu de la colonne (« `in` »), à droite (« `jn` ») ou des deux côtés (« `kn` »). Dans les trois cas, « `n` » représente le nombre par lequel il faut multiplier l'espace blanc qui serait normalement laissé sans l'un de ces spécificateurs (par défaut, la moyenne est un *em*). Ainsi, par exemple, « `|j2r|` » indiquera que nous sommes face à une colonne qui sera alignée à droite, et dans laquelle nous voulons un espace blanc d'une largeur de 1 *em*.

```
\startbuffer
\VL Colonne 1\VL Colonne 2\VL Colonne 3\VL\NR
\VL Ligne 1 \VL Texte 1.2\VL Texte 1.3\VL\NR
\VL Ligne 2 \VL Texte 2.2\VL Texte 2.3\VL\NR
\stopbuffer

\starttabulate[|i2l|j2l|k2l|]
\getbuffer
\stoptabulate
\starttabulate[|l|j2l|k2l|]
\getbuffer
\stoptabulate
\starttabulate[|i2l|l|k2l|]
\getbuffer
\stoptabulate
\starttabulate[|i2l|j2l|l|]
\getbuffer
\stoptabulate
```

Colonne 1	Colonne 2	Colonne 3
Ligne 1	Texte 1.2	Texte 1.3
Ligne 2	Texte 2.2	Texte 2.3
Colonne 1	Colonne 2	Colonne 3
Ligne 1	Texte 1.2	Texte 1.3
Ligne 2	Texte 2.2	Texte 2.3
Colonne 1	Colonne 2	Colonne 3
Ligne 1	Texte 1.2	Texte 1.3
Ligne 2	Texte 2.2	Texte 2.3
Colonne 1	Colonne 2	Colonne 3
Ligne 1	Texte 1.2	Texte 1.3
Ligne 2	Texte 2.2	Texte 2.3

- *Ajout de texte avant ou après le contenu de chaque cellule d'une colonne.* Les spécificateurs `b{Text}` et `a{Text}` font en sorte que le texte entre accolades soit écrit avant (« `b` », de *before*) ou après (« `a` », de *after*) le contenu de la cellule.
- *Appliquer une commande de formatage à l'ensemble de la colonne.* Les indicateurs « `B` », « `I` », « `S` », « `R` », « `T` » que nous avons mentionnés précédemment ne couvrent pas toutes les possibilités de formatage : par exemple, il n'y a pas d'indicateur pour les petites capitales, ou pour *sans serif*, ou qui affecte la taille de la police. Avec l'indicateur « `f\Command` », nous pouvons spécifier une commande de format qui sera automatiquement appliquée à toutes les cellules d'une colonne. Par exemple, « `|lf\cap|` » permet de composer le contenu de la colonne en petites capitales.

```
\startbuffer
\VL Colonne 1\VL Colonne 2\VL Colonne 3\VL\NR
\VL Ligne 1 \VL Texte 1.2\VL Texte 1.3\VL\NR
\VL Ligne 2 \VL Texte 2.2\VL Texte 2.3\VL\NR
\stopbuffer

\starttabulate[|l{--}|l{.}l{cap}]
\getbuffer
\stoptabulate
```

Colonne 1	Colonne 2.	COLONNE 3
Ligne 1	Texte 1.2.	TEXTE 1.3
Ligne 2	Texte 2.2.	TEXTE 2.3

- Application d'une commande quelconque à toutes les cellules de la colonne. Enfin, l'indicateur « h\Command » appliquera la commande spécifiée à toutes les cellules de la colonne.

Dans le tableau 13.1, vous trouverez quelques exemples de chaînes de spécification de format de tableau.

Spécification du format	Signification
l	Génère une colonne dont la largeur est automatiquement alignée à gauche.
rB	Génère une colonne dont la largeur est automatiquement alignée à droite, et en gras.
cIm	Génère une colonne activée pour le contenu mathématique. Centré et en italique.
j4cb{---}	Cette colonne aura un contenu centré, commencera par un tiret em (—) et ajoutera 2 <i>ems</i> d'espace blanc à droite.
l p(.7\textwidth)	génère deux colonnes : la première est alignée à gauche et de largeur automatique. La seconde occupe 70% de la largeur totale de la ligne.

Tableau 13.1 Quelques exemples de la façon de spécifier le format des colonnes dans `\tabulate`

Une fois le tableau conçu, il faut en saisir le contenu. Pour expliquer comment faire, je vais commencer par décrire comment remplir un tableau dans lequel des lignes séparent les lignes et les colonnes :

- Nous commençons par dessiner une ligne horizontale. Dans un tableau, cela se fait avec la commande `\HL` (à partir de *Horizontal Line*).
- Ensuite, nous écrivons la première ligne : au début de chaque cellule, nous voulons indiquer qu'une nouvelle cellule commence et qu'une ligne verticale doit être tracée. Cela se fait avec la commande `\VL` (à partir de *Vertical Line*). Nous commençons donc avec cette commande, et nous écrivons le contenu de chaque cellule. Chaque fois que nous changeons de cellule, nous répétons la commande `\VL`.

- À la fin d'une ligne, nous indiquons expressément qu'une nouvelle ligne va être commencée avec la commande `\NR` (de *Next Row*). Ensuite, nous répétons la commande `\HL` pour tracer une nouvelle ligne horizontale.
- Et ainsi, une par une, nous écrivons toutes les lignes du tableau. Lorsque nous avons terminé, nous ajoutons, en supplément, une commande `\NR` et une autre `\HL` pour fermer la grille avec la ligne horizontale inférieure.

Si nous ne voulons pas dessiner la grille de la table, nous supprimons les commandes `\HL` et remplaçons les commandes `\VL` par `\NC` (de l'option *New Column*).

```
\starttabulate[|1|1|1|]
\HL
\VL Colonne 1\VL Colonne 2\VL Colonne 3\VL\NR
\HL
\VL Ligne 1 \VL Texte 1.2\VL Texte 1.3\VL\NR
\VL Ligne 2 \VL Texte 2.2\VL Texte 2.3\VL\NR
\HL
\stoptabulate
```

Colonne 1	Colonne 2	Colonne 3
Ligne 1	Texte 1.2	Texte 1.3
Ligne 2	Texte 2.2	Texte 2.3

```
\starttabulate[|1|1|1|]
\NC Colonne 1\NC Colonne 2\NC Colonne 3\NC\NR
\HL
\TB
\NC Ligne 1 \NC Texte 1.2\NC Texte 1.3\NC\NR
\NC Ligne 2 \NC Texte 2.2\NC Texte 2.3\NC\NR
\stoptabulate
```

Colonne 1	Colonne 2	Colonne 3
Ligne 1	Texte 1.2	Texte 1.3
Ligne 2	Texte 2.2	Texte 2.3

Ce n'est pas particulièrement difficile lorsque l'on s'y habitue, même si lorsque l'on regarde le code source d'une table, il est difficile de se faire une idée de ce à quoi elle ressemblera. Dans `table ??`, nous voyons les commandes qui peuvent (et doivent) être utilisées dans un tableau. Il y en a certaines que je n'ai pas expliquées, mais je pense que la description que j'ai donnée est suffisante.

Et maintenant, à titre d'exemple, je vais transcrire le code avec lequel la `table ??` suivante a été écrite.

```

\placetable
[here,force]
[tbl:tablecommands]
{Commandes permettant de définir le contenu
d'un tableau}
{\starttabulate[|lT|p|]
\HL
\NC {\bf Commande}
\NC {\bf Signification}
\NR
\HL
\NC \tex{HL}
\NC Insère une ligne horizontale
\NR
\NC \tex{NR}
\NC Commence une nouvelle ligne
\NR
\NC \tex{NC}
\NC Commence une nouvelle colonne
\NR
\NC \tex{VL}
\NC Commence une nouvelle colonne en insérant
une ligne verticale de séparation
\NR
\NC \tex{EQ}
\NC Commence une nouvelle colonne en insérant
un symbole (\quotation{:} par défaut modifiable
avec \tex{setuptabulate[EQ={texte}]})
\NR
\NC \tex{NN}
\NC Commence une colonne en mode mathématique
\NR
\NC \tex{TB}
\NC Ajoute un espace vertical supplémentaire
entre deux lignes
\NR
\NC \tex{NB}
\NC Indique que la ligne suivante commence
un bloc indivisible dans lequel il ne peut y
avoir de saut de page.
\NR
\HL
\stoptabulate

```

Commande	Signification
\HL	Insère une ligne horizontale
\NR	Commence une nouvelle ligne
\NC	Commence une nouvelle colonne
\VL	Commence une nouvelle colonne en insérant une ligne verticale de séparation
\EQ	Commence une nouvelle colonne en insérant un symbole (“:” par défaut modifiable avec \setuptabulate[EQ={texte}])
\NN	Commence une colonne en mode mathéma- tique
\TB	Ajoute un espace vertical supplémentaire entre deux lignes
\NB	Indique que la ligne suivante commence un bloc indivisible dans lequel il ne peut y avoir de saut de page.

Table 1 Commandes permettant de définir le contenu d'un tableau

Le lecteur remarquera qu'en général, j'ai utilisé une (voire deux) lignes de texte pour chaque cellule. Dans un vrai fichier source, je n'aurais utilisé qu'une ligne de texte pour chaque cellule ; dans l'exemple, j'ai séparé les lignes trop longues. L'utilisation d'une seule ligne par cellule me facilite l'écriture du tableau car ce que je fais, c'est écrire le contenu de chaque cellule, sans commandes de séparation des lignes ou des colonnes. Lorsque tout est écrit, je sélectionne le texte du tableau et je demande à mon éditeur de texte d'insérer « \NC » au début de chaque ligne. Ensuite, toutes les deux lignes (car le tableau a deux colonnes), j'insère une ligne qui

ajoute la commande `\NR`, car toutes les deux colonnes commencent une nouvelle ligne. Enfin, à la main, j'insère les commandes `\HL` aux endroits où je veux qu'une ligne horizontale apparaisse. Il me faut presque plus de temps pour le décrire que pour le faire !

Mais voyez aussi comment, dans un tableau, nous pouvons utiliser les commandes ordinaires de ConTEXt. En particulier, dans ce tableau, nous utilisons continuellement `\tex` qui est expliqué dans section 10.2.4.

Pour finir, l'environnement tabulate présente les commandes habituelles pour en définir des variantes, avec `\definetabulate`, et les configurer, avec `\setuptabulate`.

```
\definetabulate [MonBeauTableau] [|lk1B|lk1|]
\setuptabulate
  [MonBeauTableau]
  [rulethickness=1mm, % épaisseur des filets
   rulecolor=darkred, % couleur des filets
   distance=1em,      % distance filets / texte
   EQ={:,:},]
\startMonBeauTableau
\HL
\NC Ligne 1 \EQ Texte 1.2\NC Texte 1.3\NC\NR
\NC Ligne 2 \EQ Texte 2.2\NC Texte 2.3\NC\NR
\HL
\stopMonBeauTableau
```

Ligne 1 : Texte 1.2 Texte 1.3
Ligne 2 : Texte 2.2 Texte 2.3

13.3.3 Tableaux avec l'environnement TABLE (tableaux naturels)

Cette partie provient en grande partie des *Fiches à Bébert*.

Les tableaux naturels offrent une configuration plus fine et une robustesse plus grande, parfois au prix d'une plus grande rigueur dans la définition.

A. Décrire le contenu du tableau

Toutes les éléments d'un tableau naturel sont des environnements, ils commencent tous par un bTRUC (en fait beginTRUC) et s'achève par un eTRUC (endTRUC). Attention les majuscules sont essentielles.

Élément délimité	Commande début	Commande fin	Commentaire
Tableau	\bTABLE	\eTABLE	
Colonne	\bTR	\eTR	Table Row
Cellule	\bTD	\eTD	Table Data
Cellule d'en-tête	\bTH	\eTH	
En-tête de la table	\bTABLEhead	\eTABLEhead	
Second en-tête de la table	\bTABLEnext	\eTABLEnext	utilisé après un changement de page
Corps de la table	\bTABLEbody	\eTABLEbody	
Fin de la table	\bTABLEfoot	\eTABLEfoot	

Tableau M.2 Commandes permettant de définir le contenu d'un tableau naturel

```
\bTABLE
\bTABLEhead
\bTR \bTH Français      \eTH \bTH Chti          \eTH
      \bTH Anglais       \eTH \bTH Italien        \eTH \eTR
\eTABLEhead
\bTABLEbody
\bTR \bTD Pantalon     \eTD \bTD Marrone      \eTD
      \bTD Pants         \eTD \bTD Pantaloni    \eTD \eTR
\bTR \bTD Serpillière   \eTD \bTD Wassingue    \eTD
      \bTD Swab           \eTD \bTD Strofinaccio \eTD \eTR
\bTR \bTD Boue          \eTD \bTD Berdoule     \eTD
      \bTD Mud            \eTD \bTD Fangos       \eTD \eTR
\eTABLEbody
\eTABLE
```

Français	Chti	Anglais	Italien
Pantalon	Marrone	Pants	Pantaloni
Serpillièvre	Wassingue	Swab	Strofinaccio
Boue	Berdoule	Mud	Fangos

B. Fusionner des cellules

Chaque cellule peut prendre les paramètres `nc` et `nr` auquels nous affectons des valeurs entières qui indiquent le nombre de cellule fusionnées, respectivement horizontalement (`cell`) et verticalement (`row`). Voyez plutôt :

```

\bTABLE
\bTR \bTD Colonne 1 Ligne 1 \eTD \bTD C2 L1 \eTD \bTD C3 L1 \eTD \eTR
\bTR \bTD Colonne 1 Ligne 2 \eTD \bTD C2 L2 \eTD \bTD C3 L2 \eTD \eTR
\bTR \bTD Colonne 1 Ligne 3 \eTD \bTD C2 L3 \eTD \bTD C3 L3 \eTD \eTR
\eTABLE
\blank[big]
\bTABLE
\bTR \bTD Colonne 1 Ligne 1 \eTD \bTD[nc=2] C2 L1 et C3 L1 \eTD \eTR
\bTR \bTD Colonne 1 Ligne 2 \eTD \bTD C2 L2 \eTD \bTD[nr=2] C3 L2, C3 L3 \eTD \eTR
\bTR \bTD Colonne 1 Ligne 3 \eTD \bTD C2 L3 \eTD \eTR
\eTABLE
\blank[big]
\bTABLE
\bTR \bTD Colonne 1 Ligne 1 \eTD \bTD C2 L1 \eTD \bTD C3 L1 \eTD \eTR
\bTR \bTD Colonne 1 Ligne 2 \eTD \bTD[nr=2,nc=2] C2 L2, C3 L2, C2 L3, C3 L3 \eTD \eTR
\bTR \bTD Colonne 1 Ligne 3 \eTD \eTR
\eTABLE

```

Colonne 1 Ligne 1	C2 L1	C3 L1
Colonne 1 Ligne 2	C2 L2	C3 L2
Colonne 1 Ligne 3	C2 L3	C3 L3

Colonne 1 Ligne 1	C2 L1 et C3 L1
Colonne 1 Ligne 2	C2 L2, C3 L2, C3 L3
Colonne 1 Ligne 3	C2 L3

Colonne 1 Ligne 1	C2 L1	C3 L1
Colonne 1 Ligne 2	C2 L2, C3	
Colonne 1 Ligne 3	L2, C2 L3, C3 L3	

C. Configurer le tableau : préciser les éléments à configurer

La configuration d'un tableau se fait à plusieurs niveaux. Il est possible de configurer l'ensemble du tableau, une seule ou un ensemble de lignes, une seule ou un ensemble de colonnes, ou bien une seule ou un ensemble de cellules. Certaines options s'appliquent à l'ensemble du tableau, d'autre à des éléments particuliers. La commande clé est `\setuptable`.

Attention, son positionnement dans le code source impacte son effet. Avant `\bTABLE`, elle s'appliquera sur la table dans son ensemble, après elle s'appliquera aux lignes, aux colonnes et aux cellules du tableau.

Pour information `option=stretch` fait en sorte que le tableau occupera l'ensemble de la largeur disponible.

```
\setupTABLE[option=stretch,color=darkred,rulethickness=3pt]
\bTABLE
\bTR \bTH A \eTH \bTH B \eTH \bTH C \eTH \bTH D \eTH \eTR
\bTR \bTD 1 \eTD \bTD 2 \eTD \bTD 3 \eTD \bTD 4 \eTD \eTR
\bTR \bTD 5 \eTD \bTD 6 \eTD \bTD 7 \eTD \bTD 8 \eTD \eTR
\eTABLE
```

A	B	C	D
1	2	3	4
5	6	7	8

```
\bTABLE
\setupTABLE[option=stretch,color=darkred,rulethickness=3pt]
\bTR \bTH A \eTH \bTH B \eTH \bTH C \eTH \bTH D \eTH \eTR
\bTR \bTD 1 \eTD \bTD 2 \eTD \bTD 3 \eTD \bTD 4 \eTD \eTR
\bTR \bTD 5 \eTD \bTD 6 \eTD \bTD 7 \eTD \bTD 8 \eTD \eTR
\eTABLE
```

A	B	C	D
1	2	3	4
5	6	7	8

Le paramétrage peut être passé directement à la commande de création d'environnement d'une ligne ou bien d'une cellule :

```
\bTABLE
\bTR \bTH A \eTH \bTH B \eTH \bTH C \eTH \bTH D \eTH \eTR
\bTR[color=darkred]
    \bTD 1 \eTD \bTD 2 \eTD \bTD 3 \eTD \bTD 4 \eTD \eTR
\bTR \bTD 5 \eTD \bTD[color=darkgreen] 6
    \eTD \bTD 7 \eTD \bTD 8 \eTD \eTR
\eTABLE
```

A	B	C	D
1	2	3	4
5	6	7	8

\setupable permet aussi de configurer des lignes et des colonnes particulières

- le premier argument permet justement d'indiquer si l'on souhaite configurer des lignes r (row) ou des colonnes c (column)

```
\setupTABLE[row]    [n] [option1,option2,...]
\setupTABLE[column] [n] [option1,option2,...]
ou
\setupTABLE[r]     [n] [option1,option2,...]
\setupTABLE[c]     [n] [option1,option2,...]
```

2. Le second argument peut prendre plusieurs valeurs.
 - **n** : Un entier indiquant le numéro de la ligne ou de la colonne que l'on souhaite modifier, ou bien une liste d'entier séparés par une virgule si l'on souhaite en modifier plusieurs : [3] modifie la troisième ligne / colonne et [2, 7, 8] affecte les deuxième, septième et huitième.
 - **first** : modifie la première ligne.
 - **last** : modifie la dernière ligne.
 - **odd** : modifie toutes les lignes impaires.
 - **even** : modifie toutes les lignes paires.
 - **each** : modifie toutes les lignes.

Et pour finir **\setupable** permet aussi de configurer des cellules particulières

```
\setupTABLE[numéro de colonne] [numéro de ligne] [option1,option2]
```

```
\bTABLE
\setupTABLE [c] [last] [color=darkgreen]
\setupTABLE [r] [2,3] [color=darkred]
\setupTABLE [4] [4] [color=darkmagenta]
\bTR \bTD 1.1 \eTD \bTD 1.2 \eTD \bTD 1.3 \eTD \bTD 1.4 \eTD \eTR
\bTR \bTD 2.1 \eTD \bTD 2.2 \eTD \bTD 2.3 \eTD \bTD 2.4 \eTD \eTR
\bTR \bTD 3.1 \eTD \bTD 3.2 \eTD \bTD 3.3 \eTD \bTD 3.4 \eTD \eTR
\bTR \bTD 4.1 \eTD \bTD 4.2 \eTD \bTD 4.3 \eTD \bTD 4.4 \eTD \eTR
\eTABLE
```

1.1	1.2	1.3	1.4
2.1	2.2	2.3	2.4
3.1	3.2	3.3	3.4
4.1	4.2	4.3	4.4

D. Configurer le tableau : les options de configuration

Les possibilités de configuration sont très nombreuses et sont assez proche de **\setupframed** car chaque environnement d'un tableau naturel se comporte de façon similaire à une **\framed** (voir section ??).

1. **align** alignement du texte. Les 4 options peuvent être combinées en les entourant d'accolades et en les séparant par des virgules.
 - * **flushleft** pour aligner le texte à gauche
 - * **middle** pour aligner le texte au centre
 - * **flushright** pour aligner à droite
 - * **inner,outer** pour aligner vers la marge interne ou externe.
 - * **lohi,high,low** permet de centrer verticalement le contenu de la cellule.

```
\bTABLE
\setupTABLE [2] [2] [color=darkmagenta,align={middle,lohi}]
\setupTABLE [c] [last] [color=darkcyan,align={flushright,bottom}]
\bTR \bTD Texte 1.1 \eTD \bTD Texte 1.2 \eTD \bTD Texte 1.3 \eTD \bTD Texte 1.4 \eTD \eTR
\bTR \bTD Texte 2.1 \eTD \bTD Texte 2.2 \eTD \bTD Texte 2.3 \eTD \bTD Texte 2.4 \eTD \eTR
\bTR \bTD Texte 3.1 \eTD \bTD Texte 3.2 \eTD \bTD Texte 3.3 \eTD \bTD Texte 3.4 \eTD \eTR
\bTR \bTD Texte 4.1 \eTD \bTD Texte 4.2 \eTD \bTD Texte 4.3 \eTD \bTD Texte 4.4 \eTD \eTR
\end{bTABLE}
```

Texte 1.1	Texte 1.2	Texte 1.3	Texte 1.4
Texte 2.1	Texte 2.2	Texte 2.3	Texte 2.4
Texte 3.1	Texte 3.2	Texte 3.3	Texte 3.4
Texte 4.1	Texte 4.2	Texte 4.3	Texte 4.4

2. Hauteur et largeur

- * `width=dimension` règle la largeur des colonnes.
- * `height=dimension` règle la hauteur des lignes.

```
\bTABLE
\setupTABLE [c] [each] [color=darkcyan,width=3cm]
\setupTABLE [c] [3] [color=darkgreen,width=6cm]
\setupTABLE [r] [2] [color=darkmagenta,height=1cm,align={middle,high}]
\bTR \bTD Texte 1.1 \eTD \bTD Texte 1.2 \eTD \bTD Texte 1.3 \eTD \bTD Texte 1.4 \eTD \eTR
\bTR \bTD Texte 2.1 \eTD \bTD Texte 2.2 \eTD \bTD Texte 2.3 \eTD \bTD Texte 2.4 \eTD \eTR
\bTR \bTD Texte 3.1 \eTD \bTD Texte 3.2 \eTD \bTD Texte 3.3 \eTD \bTD Texte 3.4 \eTD \eTR
\bTR \bTD Texte 4.1 \eTD \bTD Texte 4.2 \eTD \bTD Texte 4.3 \eTD \bTD Texte 4.4 \eTD \eTR
\end{bTABLE}
```

Texte 1.1	Texte 1.2	Texte 1.3	Texte 1.4
Texte 2.1	Texte 2.2	Texte 2.3	Texte 2.4
Texte 3.1	Texte 3.2	Texte 3.3	Texte 3.4
Texte 4.1	Texte 4.2	Texte 4.3	Texte 4.4

3. Filets (ou traits)

Il est possible de sélectionner quelle partie du cadre d'une cellule, d'une ligne ou d'une colonne nous souhaitons afficher.

- * `frame=on/off` : par défaut frame vaut on et donc un cadre entoure la cellule.
- * `topframe=on/off` : dessine ou non le trait du haut de la cellule.
- * `bottomframe=on/off` : dessine ou non le trait du bas de la cellule.
- * `leftframe=on/off` : dessine ou non le trait de gauche de la cellule.
- * `rightframe=on/off` : dessine ou non le trait de droite de la cellule.
- * `rulethickness=dimension` : l'épaisseur des traits entourant la cellule.
- * Pour pouvoir utiliser `topframe`, `bottomframe`, `leftframe` et `rightframe` il faut au préalable mettre `frame=off`.

```
\bTABLE
\setupTABLE [frame=off]
\setupTABLE [r] [1] [bottomframe=on]
\setupTABLE [c] [1] [rightframe=on]
\setupTABLE [4] [4] [frame=on,rulethickness=2pt,style=bold]
\bTR \bTD Texte 1.1 \eTD \bTD Texte 1.2 \eTD \bTD Texte 1.3 \eTD \bTD Texte 1.4 \eTD \eTR
\bTR \bTD Texte 2.1 \eTD \bTD Texte 2.2 \eTD \bTD Texte 2.3 \eTD \bTD Texte 2.4 \eTD \eTR
\bTR \bTD Texte 3.1 \eTD \bTD Texte 3.2 \eTD \bTD Texte 3.3 \eTD \bTD Texte 3.4 \eTD \eTR
\bTR \bTD Texte 4.1 \eTD \bTD Texte 4.2 \eTD \bTD Texte 4.3 \eTD \bTD Texte 4.4 \eTD \eTR
\end{bTABLE}
```

Texte 1.1	Texte 1.2	Texte 1.3	Texte 1.4
Texte 2.1	Texte 2.2	Texte 2.3	Texte 2.4
Texte 3.1	Texte 3.2	Texte 3.3	Texte 3.4
Texte 4.1	Texte 4.2	Texte 4.3	Texte 4.4

4. Style et couleurs, les options de style et de couleur sont :

- * **color=nom de la couleur** : colorie le texte et le cadre ;
- * **foregroundcolor=nom de la couleur** : colorie le texte ;
- * **background=color, backgroundcolor=nom de la couleur** : Attention c'est en deux temps, le mot color indique que l'on veut utiliser backgroundcolor, et backgroundcolor indique la couleur elle-même. On verra plus tard que background peut prendre d'autre valeur ;
- * **framecolor=nom de la couleur** : la couleur des filets.
- * **style=commande de style** : la couleur des filets.

```
\bTABLE
\setupTABLE [frame=off]
\setupTABLE [r] [1] [color=darkred, frame=on,style=\it]
\setupTABLE [2] [2] [foregroundcolor=darkcyan, frame=on]
\setupTABLE [3] [3] [background=color, backgroundcolor=magenta, frame=on]
\setupTABLE [4] [4] [framecolor=darkgreen, frame=on,style=\bf]
\bTR \bTD Texte 1.1 \eTD \bTD Texte 1.2 \eTD \bTD Texte 1.3 \eTD \bTD Texte 1.4 \eTD \eTR
\bTR \bTD Texte 2.1 \eTD \bTD Texte 2.2 \eTD \bTD Texte 2.3 \eTD \bTD Texte 2.4 \eTD \eTR
\bTR \bTD Texte 3.1 \eTD \bTD Texte 3.2 \eTD \bTD Texte 3.3 \eTD \bTD Texte 3.4 \eTD \eTR
\bTR \bTD Texte 4.1 \eTD \bTD Texte 4.2 \eTD \bTD Texte 4.3 \eTD \bTD Texte 4.4 \eTD \eTR
\end{bTABLE}
```

Texte 1.1	Texte 1.2	Texte 1.3	Texte 1.4
Texte 2.1	Texte 2.2	Texte 2.3	Texte 2.4
Texte 3.1	Texte 3.2	Texte 3.3	Texte 3.4
Texte 4.1	Texte 4.2	Texte 4.3	Texte 4.4

5. Distance et marge

- * **distance=dimension** indique la distance entre la colonne de la sélection et la suivante.
- * **leftmargindistance** et **rightmargindistance** indiquent les marges à considérer à droite et à gauche
- * **spaceinbetween=dimension** indique la distance entre deux lignes, cela s'applique à tout le tableau et doit être indiqué directement comme premier argument à **\setupTABLE**.

```
\framed[offset=none,framecolor=red]{
\setupTABLE [spaceinbetween=2mm]
\setupTABLE [distance=1cm, leftmargindistance=1cm, rightmargindistance=2cm]
\setupTABLE [c] [2] [distance=2cm]
\bTABLE
\bTR \bTD Texte 1.1 \eTD \bTD Texte 1.2 \eTD \bTD Texte 1.3 \eTD \bTD Texte 1.4 \eTD \eTR
\bTR \bTD Texte 2.1 \eTD \bTD Texte 2.2 \eTD \bTD Texte 2.3 \eTD \bTD Texte 2.4 \eTD \eTR
\bTR \bTD Texte 3.1 \eTD \bTD Texte 3.2 \eTD \bTD Texte 3.3 \eTD \bTD Texte 3.4 \eTD \eTR
\bTR \bTD Texte 4.1 \eTD \bTD Texte 4.2 \eTD \bTD Texte 4.3 \eTD \bTD Texte 4.4 \eTD \eTR
\eTABLE}
```

Texte 1.1	Texte 1.2	Texte 1.3	Texte 1.4
Texte 2.1	Texte 2.2	Texte 2.3	Texte 2.4
Texte 3.1	Texte 3.2	Texte 3.3	Texte 3.4
Texte 4.1	Texte 4.2	Texte 4.3	Texte 4.4

6. **loffset,boffset,roffset,toffset** indique la marge à gauche, bas, droite, haut des cellules de la sélection.

```
\setupTABLE [frame=off]
\setupTABLE [r] [2] [frame=on,toffset=5mm]
\setupTABLE [c] [2] [frame=on,loffset=5mm]
\bTABLE
\bTR \bTD Texte 1.1 \eTD \bTD Texte 1.2 \eTD \bTD Texte 1.3 \eTD \bTD Texte 1.4 \eTD \eTR
\bTR \bTD Texte 2.1 \eTD \bTD Texte 2.2 \eTD \bTD Texte 2.3 \eTD \bTD Texte 2.4 \eTD \eTR
\bTR \bTD Texte 3.1 \eTD \bTD Texte 3.2 \eTD \bTD Texte 3.3 \eTD \bTD Texte 3.4 \eTD \eTR
\bTR \bTD Texte 4.1 \eTD \bTD Texte 4.2 \eTD \bTD Texte 4.3 \eTD \bTD Texte 4.4 \eTD \eTR
\eTABLE
```

Texte 1.1	Texte 1.2	Texte 1.3	Texte 1.4
Texte 2.1	Texte 2.2	Texte 2.3	Texte 2.4
Texte 3.1	Texte 3.2	Texte 3.3	Texte 3.4
Texte 4.1	Texte 4.2	Texte 4.3	Texte 4.4

E. Quelques exemples

Il est possible de ranger tous les éléments de configuration dans un « `setup` » et d'y faire appel ensuite. Regardez :

```
\startsetups SetupMaTable
\setupTABLE [frame=off,framecolor=darkred,option=stretch,
             offset=1mm,align=flushright,rulethickness=2pt]
\setupTABLE [r] [first]
             [foregroundcolor=darkred,style={\ss\bf},
              bottomframe=on,rulethickness=1pt]
\setupTABLE [1] [1] [bottomframe=off]
\setupTABLE [c] [first]
             [style={\ss\bf}, loffset=5mm, width=3cm,align=flushleft]
\setupTABLE [r] [last]
             [bottomframe=on]
\stopsetups

\bTABLE[option=stretch]
\setups{SetupMaTable}
\bTR \bTD \eTD
\bTD France \eTD \bTD Royaume-Uni \eTD
\bTD Suède \eTD \bTD Allemagne \eTD
\eTR
\bTR \bTD Capitale \eTD
\bTD Paris \eTD \bTD Londres \eTD
\bTD Stockholm \eTD \bTD Berlin \eTD
\eTR
\bTR \bTD Population \eTD
\bTD $67\,422\,241\$ \eTD \bTD $66\,465\,641\$ \eTD
\bTD $10\,333\,456\$ \eTD \bTD $83\,042\,235\$ \eTD
\eTR
\endTABLE
```

	France	Royaume-Uni	Suède	Allemagne
Capitale	Paris	Londres	Stockholm	Berlin
Population	67 422 241	66 465 641	10 333 456	83 042 235

Un autre exemple pour aligner les chiffres.

```
\bTABLE
\setupTABLE[c][1][align=right]
\setupTABLE[c][2][aligncharacter=yes,alignmentcharacter={.},align=middle]
\setupTABLE[c][3][aligncharacter=yes,alignmentcharacter={.},align=middle]
\bTR \bTH Categorie \eTH \bTH Valeur A \eTH \bTH Valeur B \eTH \eTR
\bTR \bTD Premier \eTD \bTD $71.35$ \eTD \bTD 1.00\% \eTD \eTR
\bTR \bTD Seconde \eTD \bTD $43.7$ \eTD \bTD 10.0\% \eTD \eTR
\bTR \bTD Total \eTD \bTD $115.0$ \eTD \bTD 25.0\% \eTD \eTR
\eTABLE
```

Categorie	Valeur A	Valeur B
Premier	71.35	1.00%
Seconde	43.7	10.0 %
Total	115.0	25.0 %

13.4 Aspects communs aux images, tableaux et autres objets flottants

Nous savons déjà que les images et les tableaux ne sont pas obligatoirement des objets flottants, mais ils sont de bons candidats pour l'être, et il faut alors utiliser les commandes `\placefigure` ou `\placetable`. En plus de ces deux commandes, et avec la même structure, dans ConTEXt nous avons la commande `\placechemical` (pour insérer des formules chimiques), la commande `\placegraphic` (pour insérer des graphiques) et la commande `\placeintermezzo` pour insérer une structure que ConTEXt appelle *Intermezzo* et que je soupçonne de faire référence à des fragments de texte encadrés. Toutes ces commandes sont à leur tour des applications concrètes d'une commande plus générale qui est `\placefloat` dont la syntaxe est la suivante

```
\placefloat [Nom] [Options] [Étiquette] {Titre} {Contenu}
```

Notez que `\placefloat` est identique à `\placefigure` et `\placetable` à l'exception du premier argument qui dans `\placefloat` prend le nom de l'objet flottant. En effet, *chaque type d'objet flottant peut être inséré dans le document avec deux commandes différentes : \placefloat[TypeName] ou \placeTypeName*. En d'autres termes : `\placefloat[figure]` et `\placefigure` sont exactement la même commande, tout comme `\placefloat[table]` est la même commande que `\placetable`.

Je parlerai donc désormais de `\placefloat`, mais sachez que tout ce que je dirai s'appliquera également à `\placefigure` ou `\placetable` qui sont des applications spécifiques de cette commande.

Les arguments `\placefloat` sont :

- *Nom*. fait référence à l'objet flottant en question. Il peut s'agir d'un objet flottant prédéterminé (`figure`, `tableau`, `chimique`, `intermezzo`) ou d'un objet flottant créé par nos soins à l'aide de `\definefloat` (voir section 13.5).
- *Options*. Une série de mots symboliques qui indiquent à ConTEXt la façon dont il doit insérer l'objet. La grande majorité d'entre eux font référence à où l'insérer. Nous verrons cela dans la section suivante.
- *Étiquette*. Une étiquette pour les futures références internes à cet objet.
- *Titre*. Le texte du titre à ajouter à l'objet. Concernant sa configuration, voir section 13.4.2.
- *Contenu*. Cela dépend, bien entendu, du type d'objet. Pour les images, il s'agit généralement d'une commande `\externalimage` ; pour les tableaux, des commandes qui permettront de créer le tableau ; pour les *intermezzi*, d'un fragment de texte encadré ; etc.

Les trois premiers arguments, qui sont introduits entre crochets, sont facultatifs. Les deux derniers (qui sont introduits entre crochets) sont obligatoires, bien qu'ils puissent être vides. Ainsi, par exemple :

```
\placefloat{}{}
```

undefined

Figure 1

Note : Nous voyons que ConTeXt a considéré que l'objet à insérer était une image, puisqu'il a été numéroté comme une image et inclus dans la liste des images. Cela me fait supposer que les objets flottants sont des images par défaut.



13.4.1 Options d'insertion d'objets flottants

L'argument *Options* dans `\placefigure`, `\placetable` et `\placefloat` contrôle différents aspects concernant l'insertion de ces types d'objets. Il s'agit principalement de l'endroit de la page où l'objet sera inséré. Ici, plusieurs valeurs sont prises en charge, chacune d'une nature différente :

- Certains des emplacements d'insertion sont établis par rapport à des éléments de la page (`top`, `bottom` `inleft`, `inright`, `inmargin`, `margin`, `leftmargin`, `rightmargin`, `leftedge`, `rightedge`, `innermargin`, `inneredge`, `outeredge`, `inner`, `outer`). Il doit, bien entendu, s'agir d'un objet qui peut tenir dans la zone où il est destiné à être placé et un espace doit avoir été réservé pour cet élément dans la mise en page. À ce sujet, voir la section 5.2 et 5.3.
- Les autres emplacements d'insertion possibles sont davantage liés au texte entourant l'objet et indiquent où l'objet doit être placé pour que le texte circule autour de lui. Il s'agit essentiellement des valeurs `left` et `right`.

```
\useMPlibrary [dum]
\setupfloat[figure][location=]
\placefigure [left,none] []
{ }
{\externalfigure [dummy]
[height=1cm,width=3cm]}
Ceci est un petit texte pour voir ce que donne l'insection d'une petite
image. Ceci est un petit texte pour voir ce que donne l'insection d'une
petite image.
```

Ceci est un petit texte pour voir ce que donne l'insection d'une petite image. Ceci est un petit texte pour voir ce que donne l'insection d'une petite image.
state: unknown

- L'option `here` est interprétée comme une recommandation de conserver l'objet à l'endroit du fichier source où il se trouve. Cette *recommandation* ne sera pas respectée si les exigences de pagination ne le permettent pas. Cette indication est renforcée si nous ajoutons l'option `force` qui signifie exactement cela : forcer l'insertion de l'objet à cet endroit. Notez qu'en forçant l'insertion à un point particulier, la séquence des éléments du code source sera la même que celle du document final.
- Les autres options possibles concernent la page sur laquelle l'objet doit être inséré : `page` l'insère sur une nouvelle page ; `opposite` l'insère sur la page opposée à la page actuelle ; `leftpage` sur une page paire ; `rightpage` sur une page impaire.

Il existe certaines options qui ne sont pas liées à l'emplacement de l'objet. Parmi elles :

- `none` : Cette option permet de supprimer le titre.
- `split` : Cette option permet à l'objet de s'étendre sur plus d'une page. Il doit, bien entendu, s'agir d'un objet divisible par nature, comme un tableau. Lorsque cette option est utilisée et que l'objet est divisé, on ne peut plus dire qu'il est flottant.

13.4.2 Configuration des titres des objets flottants

À moins d'utiliser l'option « `none` » dans `\placefloat`, par défaut, les objets flottants sont associés à un titre composé de trois éléments :

- Le nom du type d'objet en question. Ce nom est exactement celui du type d'objet ; ainsi, si, par exemple, nous définissons un nouvel objet flottant appelé « Séquence » et que nous insérons une « Séquence » comme objet flottant, le titre sera « Séquence 1 ». Il est aussi possible d'utiliser la commande `\setuplabeltext` (cf. section 10.5.3) :

```
\mainlanguage[fr]
\definefloat[Séquence][Séquences]
\placeSéquence[]{-}

\setuplabeltext[fr][Séquence=Séq.~]
\placeSéquence[]{-}
```

undefined

Séquence 1

undefined

Séq. 2

Malgré ce qui vient d'être dit, si la langue principale du document n'est pas l'anglais, le nom anglais des objets prédéfinis, comme par exemple les objets « `figure` » ou « `table` », sera traduit ; Ainsi, par exemple, l'objet « `figure` » dans les documents en français est appelé « `Figure` », tandis que l'objet « `table` » esappelé « `Table` ». Ces noms français pour les objets prédéfinis peuvent être modifiés avec `\setuplabeltext` comme expliqué dans [section 10.5.3](#).

- Son numéro. Par défaut, les objets sont numérotés par chapitre, ainsi le premier tableau du chapitre 3 sera donc le tableau « 3.1 ».
- Son contenu. Introduit comme argument de `\placefloat`.

Avec `\setupcaptions` ou `\setupcaption[TypeDeFlottant]`, nous pouvons modifier le système de numérotation et l'apparence du titre lui-même. La première commande affectera tous les titres de tous les objets, et la seconde n'affectera que le titre d'un type d'objet particulier :

- comme pour le système de numérotation (voir [section B](#)), il est contrôlé par les options `number`, `way`, `prefixsegments` et `numberconversion` :
 - `number` peut adopter les valeurs `yes`, `no` ou `none` et contrôle la présence ou non d'un numéro.
 - `way` indique si la numérotation sera séquentielle dans tout le document (`way=bytext`), ou si elle recommencera au début de chaque chapitre (`way=bychapter`) ou section (`way=bysection`). Dans le cas d'un redémarrage, il convient de coordonner la valeur de cette option avec celle de l'option `prefixsegments`.
 - `prefixsegments` indique si le numéro aura un *préfixe*, et quel sera ce dernier. Ainsi, `prefixsegments=chapter` fait en sorte que le nombre d'objets commence toujours par le numéro de chapitre, tandis que `prefixsegments=section` fera précéder le numéro d'objet du numéro de section et `prefixsegments=chapter:section` combinera les deux.
 - `numberconversion` contrôle le type de numération (voir [section B](#)). Les valeurs de cette option peuvent être : des chiffres arabes (« `numbers` »), des lettres minuscules (« `a` », « `characters` »), majuscules (« `A` », « `Characters` »), petites capitales (« `KA` »), des chiffres romains minuscules (« `i` », « `r` »), « `romannumerals` » majuscules (« `I` », « `R` », « `Romannumerals` »), ou petites capitales (« `KR` »)).
- L'apparence du titre lui-même est contrôlée par de nombreuses options. Je vais les énumérer, mais pour une explication détaillée de la signification de chacune d'entre elles, je vous renvoie à [section 7.4.4](#) où est expliqué le contrôle de l'apparence des commandes de sectionnement, car les options sont largement similaires. Les options en question sont :
 - Pour contrôler le format de tous les éléments du titre, `style`, `color`, `command`.
 - Pour contrôler le format uniquement du nom du type d'objet : `headstyle`, `headcolor`, `headcommand`, `headseparator`.
 - Pour contrôler uniquement le format de la numérotation : `numbercommand`.
 - Pour contrôler uniquement le format du titre lui-même : `textcommand`.

- Sa position, avec `location`, peut adopter de très nombreuses valeurs `left`, `right`, `middle`, `low`, `lefthanging`, `righthanging`, `hang`, ..., que je vais tacher d'illustrer dans une mise à jour.
- Nous pouvons également contrôler d'autres aspects tels que la distance entre les différents éléments qui composent le titre, la largeur du titre, son placement par rapport à l'objet, etc. Je renvoie ici aux informations contenues dans [ConTeXt wiki](#) concernant les options qui peuvent être configurées avec cette commande.

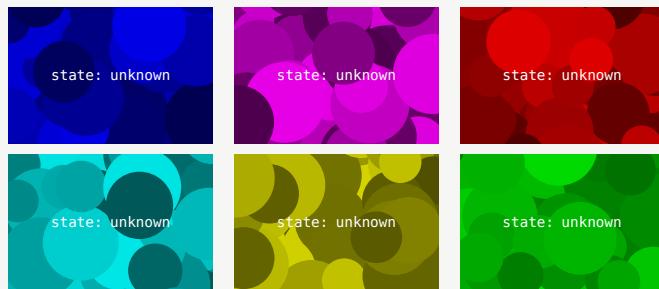
13.4.3 Insertion combinée de deux ou plusieurs objets

Pour insérer deux ou plusieurs objets différents dans le document, de telle sorte que ConTeXt les garde ensemble et les traite comme un seul objet, nous disposons de l'environnement `\startcombination` dont la syntaxe est :

```
\startcombination[ModeAffichage] ... \stopcombination
```

où *ModeAffichage* indique comment les objets doivent être ordonnés : s'ils doivent tous être ordonnés horizontalement, *ModeAffichage* indique seulement le nombre d'objets à combiner. Mais si l'on veut combiner les objets sur deux ou plusieurs lignes, il faudra indiquer le numéro de l'objet par ligne, suivi du nombre de lignes, et séparer les deux numéros par le caractère *. Par exemple :

```
\useMPlibrary [dum]
\startcombination [3*2]
  {\externalfigure [dummy] [height=2cm,width=3cm]}
  {\externalfigure [dummy] [height=2cm,width=3cm]}
\stopcombination
```



Dans l'exemple précédent, les images que j'ai combinées n'existent pas en réalité, c'est pourquoi, au lieu des images, ConTeXt a généré des images aléatoires (avec `\useMPlibrary [dum]` et `\externalfigure [dummy]`).

Voyez, d'autre part, comment chaque élément à combiner dans `\startcombination` est entre accolades. L'ensemble des accolades constitue une liste d'arguments.

En fait, `\startcombination` nous permet non seulement de connecter et d'aligner des images, mais aussi n'importe quel type de *boîte* comme des textes dans un environnement `\startframedtext`, des tableaux, etc. Pour configurer la combinaison, nous pouvons utiliser la commande `\setupcombination` et nous pouvons également créer des combinaisons préconfigurées en utilisant `\definecombination`.

13.4.4 Configuration générale des objets flottants

Nous avons déjà vu qu'avec `\placefloat` nous pouvons contrôler l'emplacement de l'objet flottant inséré et quelques autres détails. Il est également possible de configurer :

- Les caractéristiques globales d'un type particulier d'objet flottant. Pour ce faire, utilisez la commande `\setupfloat[Nom du type d'objet flottant]`.
- Les caractéristiques globales de tous les objets flottants de notre document. Pour ce faire, utilisez `\setupfloats`.

N'oubliez pas que, de la même manière que `\placefloat[figure]` est équivalent à `\placefigure`, `\setupfloat[figure]` est équivalent à `\setupfigures`, et `\setupfloat[table]` est équivalent à `\setuptables`.

En ce qui concerne les options configurables de ces dernières, je me réfère à la liste officielle des commandes de ConTeXt ([section 3.6](#)), ainsi qu'au [wiki](#)

13.5 Définition d'objets flottants supplémentaires

La commande `\definefloat` nous permet de définir nos propres objets flottants. Sa syntaxe est la suivante :

```
\definefloat [NomSingulier] [NomPluriel] [Configuration]
```

Où l'argument *Configuration* est un argument facultatif qui nous permet d'indiquer déjà la configuration de ce nouvel objet au moment de sa création. Nous pouvons également le faire plus tard avec `\setupfloat [NomSingulier]`.

Puisque nous terminons notre introduction par cette section, je vais en profiter pour approfondir un peu plus l'apparente *jungle* des commandes ConTeXt qui, une fois comprise, n'est pas tant une *jungle* mais est, en fait, tout à fait rationnelle.

Commençons par nous demander ce qu'est réellement un objet flottant pour ConTeXt, la réponse étant qu'il s'agit d'un objet ayant les caractéristiques suivantes :

- il dispose d'une certaine marge de liberté quant à son emplacement sur la page.
- il est associé à une liste qui lui permet de numérotter ce type d'objets et, éventuellement, d'en générer un index.
- il possède un titre
- lorsque l'objet peut réellement flotter, il doit être traité comme une unité indissociable, c'est-à-dire (dans la terminologie T_EX) *enfermé dans une boîte*

En d'autres termes, l'objet flottant est en fait constitué de trois éléments : l'objet lui-même, la liste qui lui est associée et le titre. Pour contrôler l'objet lui-même, nous n'avons besoin que d'une commande pour définir son emplacement et d'une autre pour insérer l'objet dans le document ; pour définir les aspects de la liste, les commandes générales de contrôle de la liste sont suffisantes, et pour définir les aspects du titre, les commandes générales de contrôle du titre.

Et c'est là qu'intervient le génie de ConTeXt : une commande simple pour contrôler les objets flottants (`\setupfloats`), et une commande simple pour insérer les objets flottants : `\placefloat`, auraient pu être conçues : mais ce que fait ConTeXt est de :

1. Conçoit une commande permettant de lier un nom à une configuration d'objet flottant spécifique. Il s'agit de la commande `\definefloat`, qui ne lie pas réellement un nom, mais deux noms, un au singulier et un au pluriel.
2. Créez, avec la commande de configuration globale des objets flottants, une commande qui nous permet de configurer uniquement un type d'objet spécifique : `\setupfloat [NomSingulier]`.
3. Ajouter à la commande de localisation des objets flottants, (`\placefloat`), un argument qui nous permet de différencier l'un ou l'autre type : (`\placefloat [NomSingulier]`).
4. Crée des commandes, y compris le nom de l'objet, pour toutes les actions d'un objet flottant. Certaines de ces commandes (qui sont en fait des clones d'autres commandes plus générales) utiliseront le nom de l'objet au singulier et d'autres l'utiliseront au pluriel.

Par conséquent, lorsque nous créons un nouvel objet flottant et que nous indiquons à ConTeXt son nom au singulier et au pluriel, ConTeXt :

- Réserve un espace en mémoire pour stocker la configuration spécifique de ce type d'objet.
- Crée une nouvelle liste avec le nom singulier de ce type d'objet, puisque les objets flottants sont associés à une liste.
- Crée un nouveau type de « titre » lié à ce nouveau type d'objet, afin de conserver une configuration personnalisée de ces titres.
- Et enfin, elle crée un groupe de nouvelles commandes spécifiques à ce nouveau type d'objet, dont le nom est en fait un synonyme de la commande plus générale.

Dans [table 13.3](#), nous pouvons voir les commandes qui sont automatiquement créées lorsque nous définissons un nouvel objet flottant, ainsi que les commandes plus générales dont elles sont les synonymes :

Commande	Synonyme de	Exemple
\completelistof<NomPluriel>	\completelist[NomPluriel]	\completelistoffigures
\place<NomSingulier>	\placefloat[NomSingulier]	\placefigure
\placelistof<PluralName>	\placelist[PluralName]	\placelistoffigures
\setup<NomSingulier>	\setupfloat[NomSingulier]	\setupfigure

Tableau 13.3 Commandes générées à la création d'un nouvel objet flottant

En fait, quelques commandes supplémentaires sont créées qui sont synonymes des précédentes et comme je ne les ai pas incluses dans l'explication du chapitre, je les ai omises de [table 13.3](#) : `\start<NomSingulier>`, `\start<NomSingulier>text` et `\startplace<NomSingulier>`.

J'ai utilisé la commande utilisée pour les images comme exemple des commandes créées lors de la définition d'un nouvel objet flottant ; et je l'ai fait parce que les images, comme les tableaux et le reste des objets flottants prédéfinis par ConTeXt, sont des cas réels de `\definefloat` :

```
\definefloat[chemical][chemicals]
\definefloat[figure][figures]
\definefloat[table][tables]
\definefloat[intermezzo][intermezzi]
\definefloat[graphic][graphics]
```

Enfin, nous constatons qu'en réalité, la commande ConTeXt ne contrôle en aucun cas le type de matériel inclus dans chaque objet flottant particulier ; elle présume que c'est le travail de l'auteur. C'est pourquoi nous pouvons également insérer du texte avec les commandes `\placefigure` ou `\placetable`. Toutefois, le texte saisi avec `\placefigure` est inclus dans la liste des images, et s'il est saisi avec `\placetable`, dans la liste des tableaux.

I

Appendices

Annexe A

Installing, configuring and updating ConTEXt

TeX's main distributions (TeX Live, teTeX, MikTeX, MacTeX, etc.) include a version of ConTEXt. However, this is not the most updated version. In this appendix I will explain two procedures to install two different versions of ConTEXt ; the first includes both ConTEXt Mark II and Mark IV and the second includes only ConTEXt Mark IV.

The installation procedure follows the same steps on any operating system ; but the details change from one system to another. However, we can simplify things in such a way that in the following lines I will distinguish between two big groups of systems :

- **Unix-type systems** : This includes Unix itself, as well as GNU Linux, Mac OS, FreeBSD, OpenBSD or Solaris. The procedure is basically the same in all these systems ; there are some very small differences that I will highlight in the appropriate place.
- **Windows systems**, that includes the different versions of that operating system : Windows 10 (the latest version, I think), Windows 8, Windows 7, Windows Vista, Windows XP, Windows NT, etc.

Important note on the installation process on Microsoft Windows systems :

ConTEXt, like all TeX systems, is designed to work from a terminal ; the programs and procedures for installation, too. In Windows this is also perfectly possible and should not create any major difficulty. The problem is that, on the one hand Windows users are not always used to doing this, and on the other, since Windows came into being in the *illusion* (false) that everything in a computer system could be done graphically, in general the versions of that operating system do not *advertise* too much about how to use the terminal. And then, it is common for each version of this system to change the name of the program that runs the terminal and how to open it. As far as I know, the Windows terminal emulation program has been given many names : « DOS window », « Command Prompt », « cmd », etc. The location of this program in the Windows application menu also changes depending on the version of Windows in question.

I stopped using Windows-based systems in 2004, so there is little I can do here to help the reader. He or she will have to figure out, on their own, how to open a terminal in their particular version of the operating system ; which shouldn't be too difficult.

1 Installing and configuring « ConTeXt Standalone »

The ConTeXt distribution known as « Standalone », also known as « ConTeXt Suite », is a complete and updated distribution of ConTeXt, which downloads the necessary files from the Internet, does not take up too much disk space, is easy to update, and above all — hence the name *Standalone* — is contained in a single directory which can be located anywhere we want on the hard disk. It would even be possible for a single computer to have several versions of ConTeXt each in its own directory. This distribution includes the fonts, binary files and documentation needed to run ConTeXt Mark II (which implies the TeX PdfLatex and XeTeX engines), and ConTeXt Mark IV (which implies the LuaTeX engine).

For information about TeX engines, see section ?? ; and on TeX engines in relation to ConTeXt, as well as the versions known as Mark II and Mark IV, section ??.

The following explains how to install, run, update and restore « ConTeXt Standalone » on our system. The data and procedures provided here are a summary of the much more extensive information included in the [ConTeXt wiki](#), to which I have added some additional detail drawn from a wikibook on ConTeXt hosted on [wikibooks](#). If there is any problem with the installation, or if you want to extend any detail, you should directly consult any of these (though the latter is in French)

1.1 Installation

Installing « ConTeXt Standalone » means having an Internet connection, and implies the following steps :

1. Creating the directory in which ConTeXt will be installed.
2. Downloading the installation *script* into this directory.
3. Running this *script* with the desired options.
4. Making some final adjustments.

Step 1 : creating the installation directory

This, in fact, has nothing to do with ConTeXt and we have to assume that every user will know how to do it. In Windows systems the normal way is to do it from the file manager. On Unix-type systems, it can be done from a file manager or from a terminal. It is important, however, to keep in mind that it is not recommended that the installation directory contains any blank space in your path. I personally also tend to shy away from using non-English directory names with things like accented vowels in them.

From now on I will assume that the installation directory is, in Unix-like systems, « ~/context/ » and in Windows, « C :\Programs\context ».

Step 2 : Download the installation *script* into the installation directory

The installation *script* will differ according to the operating system you are installing on :

- On Unix-like systems it can be downloaded, with a web browser, or, from a terminal with « `wget` » or « `rsync` » :

```
wget http://minimals.contextgarden.net/setup/first-setup.sh  
rsync rsync://minimals.contextgarden.net/setup/first-setup.sh
```

- On Windows-type systems, as far as I know, there are no standard tools for downloading from the console. It has to be done with a web browser. The download address can be any of the following :

```
http://minimals.contextgarden.net/setup/context-setup-mswin.zip  
http://minimals.contextgarden.net/setup/context-setup-win64.zip
```

Once downloaded, in Windows you have to unzip the file,

Step 3 : Run the installation *script*

The installation *script* must be run from the terminal. In Unix-type systems the name of the *script* is « `first-setup.sh` » and can be run with `bash` or `sh`. In Windows-type systems the *script* is called « `first-setup.bat` » and is run by simply typing its name in the system console or MS-DOS window from the installation directory.

The installation *script* allows for the following options :

- `--context` : this option determines which version of ConTeXt will be installed, whether the most recent development version (« `-context=latest` ») or the latest stable version (« `-context=beta` »). The default value is « `beta` ».
- `--engine` : allows us to indicate whether we want to install Mark IV (« `-engine=luatex` », the default value) or Mark II.
- `--modules` : also install the ConTeXt extension modules that do not belong to the distribution as such, but that offer interesting additional utilities. To do this we need to indicate « `-modules=all` ».

With regard to the installation options, I believe that the information in the wiki is now obsolete. There it says that to install only Mark IV you need to explicitly indicate the "`-engine=luatex`" option and that the "`-context=latest`" option installs the latest stable version, not the development version. However, from halfway through 2020 the content of `first-setup.sh` changed, and taking a look inside it I found that to install the very latest version you need to expressly indicate "`-context=latest`", and that "`-engine=luatex`" is enabled by default.

The French Wikibook I mentioned at the beginning adds two other possible options to the options I just mentioned (documented on the ConTeXt wiki) : « `-fonts=all` » and « `goodies=all` ». ConTeXtgarden doesn't mention them, but including them in the installation command as well doesn't hurt. Therefore I would advise you to run the installation script with the following options (depending on whether we are on a Unix- or Windows-type system) :

- Unix :`bash first-setup.sh --context=latest --modules=all --fonts=all --goodies=all`
- Windows :`first-setup.bat --context=latest --modules=all --fonts=all --goodies=all`

This, depending on the speed of our Internet connection, may take some time, but not too much.

Configuring a proxy

The installation script uses rsync to obtain the necessary files. So, if you are behind a proxy server, you need to specify its details to rsync. The easiest way to set this is to set the variable RSYNC_PROXY in the terminal or in your startup *script* (.bashrc or the corresponding file for each shell). Replace the username, password, proxyhost and proxyport with the correct information. This is done, on Unix-type systems, with the « export » command, and in Windows-type systems with the « set » command. For example :

```
export RSYNC_PROXY=username:password@proxyhost:proxyport
```

Sometimes, when we are behind a firewall, port 873 may be closed for outgoing TCP connections. If port 22 is open for ssh connections, one trick that can be used is to connect to a computer somewhere outside the firewall and tunnel into port 873 (using the nc program).

```
export RSYNC_CONNECT_PROG='ssh tunelhost nc %H 873'
```

where the « tunnelhost » is the machine outside the firewall we have access to. Of course, this machine must have nc and port 873 open for the outgoing TCP connection

After running « *first-setup* » in the installation directory two new directories will appear called, « bin » and « tex » respectively.

Step 4 : Final adjustments (Only on GNU Linux)

In GNU Linux systems there are many directories where fonts can be installed. If we want ConTeXt to use these fonts we must tell it where to find them. To do this we must add the following line to the « *tex/setuptex* » file created after the installation :

```
export OSFONTPDIR="~/.fonts:/usr/share/fonts:/usr/share/texmf/fonts/opentype/"
```

with which the environment variable OSFONTPDIR is loaded with the three directories in which the fonts installed in the system are normally located

The /usr/share/texmf/fonts/ will only be there if there is some other installation of \TeX or other systems based on it in our operating system ; in this case it should be included in the OSFONTPDIR path so we can use the opentype fonts that such an installation may have included. If you have any commercial fonts that you want ConTeXt to use, you have to make sure that the path to these is one of those included in OSFONTPDIR, or otherwise, add the path to this variable. I have seen, for example, that some fonts are installed in /usr/local/fonts instead of /usr/share/fonts.

Finally, it may be a good idea to have ConTeXt generate a database with the necessary files for execution. This will be done by running the following three commands from a terminal :

```
. ~/context/tex/setuptex
context --generate
context --make
```

The first instruction is a point (dot). That's an abbreviation for bash's internal source command. We can also, of course, run *source* if it's more convenient for us.

1.2 Running « ConTeXt Standalone »

« ConTeXt Standalone » has been designed to be able to coexist with other installations of TeX systems, which is an advantage because it allows us to have several different versions installed on the same operating system ; but in order to exploit this advantage it is essential that the environment variables needed to run ConTeXt are not set permanently, because every time we start a terminal to run « context » from it, we'll have to start by loading these environment variables into memory. They are contained in the « *tex/setuptex* » (Unix) or « *tex\setuptex.bat* » (Windows) file. This is done :

- In Unix-type systems, after opening the terminal in which we want to use « *context* », by running either of the following two commands :

```
source ~/context/tex/setuptex  
. ~/context/tex/setuptex
```

(assuming that the directory where the version of « *context* » we want to use is « *~/context* »).

- In Windows-type systems, by running the *tex\setuptex.bat* command from the installation directory in the terminal from which we will use ConTeXt.

If there is no other installation of TeX or any of its derivatives in our system, we can avoid this by automating the execution of this order every time a terminal is opened :

- On Unix-like systems this is done by including it in the file containing the general terminal startup *script* (usually « *.bashrc* »).

The configuration file of a terminal depends on the *shell* program that the terminal uses by default. If this is bash (which is the most used in GNU Linux systems), the file read at the beginning is *.bashrc*. The sh and ksh shells use a file called *.profile*, zsh uses *.zshenv*, and tcsh or csh read the *.cshrc* file. Some specific implementation may change the names of these files and so, for example, *.bashrc* is sometimes called *.bash_profile*.

- In Windows-type systems we can create a shortcut on the desktop that runs cmd.exe and then edit it, putting as a command to run when we double click on it :

```
C:\WINDOWS\System32\cmd.exe /k C:\Programs\context\tex\setuptex.bat
```

Another possibility, if we do not wish to run this script each time we want to use ConTeXt, nor want to permanently set the environment variables necessary for it to be run, is to do it from the text editor itself, instead of running ConTeXt from a terminal. How you do this depends on the particular text editor you are using. The ConTeXt wiki provides information on how to set up various common editors : LEd, Notepad++, Scite, TeXnicCenter, TeXworks, vim and some others.

1.3 Updating the version of « ConTeXt Standalone » or returning to an earlier version

Mark IV is still under development, so « ConTeXt Standalone » is often updated. To update our installation just repeat the process : we download a new version of « `first-setup.sh` » and run it.

If, for whatever reason, we want to go back to a previous version of « ConTeXt Standalone », just run « `first-setup` » with the « `--context=date` » option, where *date* is the date corresponding to the version we want to recover. Note that the date has to be introduced in the US months-days-years format.

The complete list of ConTeXt versions and associated dates can be found at [this link](#).

Finally, keep in mind that after reinstalling the system, whether it is to upgrade or to return to a previous version, on GNU Linux systems you will have to run step 4 of the installation again, which I have called « Final Adjustments ».

2 Installing LMTX

If we only plan to use ConTeXt Mark IV, and we want to compile our projects not directly with LuaTeX but with LuaMetaTeX, a simplified LuaTeX that uses less system resources and that can work on *less powerful* systems, instead of « ConTeXt Standalone », we need to install LMTX which is the latest version of ConTeXt. The name is an acronym of the name of the TeX engine being used : LuaMetaTeX. This version was launched in 2019, and since approximately May 2020 it is the recommended default ConTeXt distribution as suggested in [ConTeXt wiki](#).

The current development of LMTX is intense, and the beta version can change several times a week. Some of its developments, moreover, temporarily pose certain incompatibilities with Mark IV, and so, for example, while I am writing these lines, the latest version of LMTX (August 4, 2020) produces an error with the `\Caps` command. Therefore I would advise newcomers, for the moment, to work with « ConTeXt Standalone » instead.

2.1 The installation itself

The installation is as simple as :

- **Step 1 :** Decide on the directory you want to install LMTX in, and, if necessary, create it. I will assume that the installation is done in a directory called « context » located in our user directory.
- **Step 2 :** Download (to the installation directory) the zip file from the [ConTeXt wiki](#) that corresponds to your operating system and processor. It can be any of the following :
 - GNU/Linux
 - ★ X86 Processor
 - ▷ [32 bit version.](#)
 - ▷ [64 bit version.](#)
 - ★ ARM Processor
 - ▷ [32 bit version.](#)
 - ▷ [64 bit version.](#)
 - Microsoft Windows
 - ★ [32 bit version](#)
 - ★ [64 bit version](#)
 - Mac OS, [versión de 64 bits](#)
 - FreeBSD
 - ★ [32 bit version.](#)
 - ★ [64 bit version.](#)
 - OpenBSD6.6
 - ★ [32 bit version.](#)
 - ★ [64 bit version.](#)
 - OpenBSD6.7
 - ★ [32 bit version.](#)
 - ★ [64 bit version.](#)

If you don't know whether your system is 32-bit or 64-bit, chances are – unless your computer is very old – it's 64-bit. If you don't know whether your processor is X86 or ARM, it's most likely X86.

- **Step 3 :** Unzip, the file downloaded in the previous step into the installation directory. A folder will be created called « bin » and two files, one called « installation.pdf », that contains more detailed information about the installation, and a second file which is the actual installation program called « install.sh » (in Unix-type systems) or « install.bat » (in Windows systems).
- **Step 4 :** Run the installation program (« install.sh » or « install.bat »). It needs an Internet connection as the installation program searches the web for the files it needs.
 - On Unix-type systems the installation program, located in the installation directory, is run from a terminal, either with bash, or with sh. It is not necessary to have administrator privileges, unless the installation directory is outside the user's « home » directory.
 - In Windows-type systems, you must open a terminal, move to the installation directory, and from the terminal, run install.bat. It is not necessary here either that the installation program is run as a system administrator, but it is recommended that this be done so that symbolic links of the files can be used, thus saving disk space.

- **Step 5** inform the system of the path to LMTX :

In Windows systems, the installation program generates a file, called « set-path.bat » which updates all the configuration files necessary to let Windows know that you have installed LMTX in the system and where you has done so. In GNU Linux systems, FreeBSD or Mac OS no *script* that automates the task is generated, so we must incorporate the address for the ConTeXt binaries in the system's PATH variable, which we would get by running in the terminal, from the installation :

```
export PATH="InstallationDir/tex/texmf-Platform/bin : "$PATH
```

where *InstallationDir* is the installation directory (for example, “/home/user/context”) and *texmf-Platform* will vary according to the version of LMTX we have installed. For example, an installation on a 64 bit Linux system, *texmf-Platform* will be “texmf-linux-64”. Therefore we should run the following command from the terminal :

```
export PATH="/home/user/context/texmf-linux-64/bin : "$PATH
```

This command will include LMTX in the system path, only as long as the terminal from which it has been run remains open. If we want this to be done automatically every time a terminal is opened, we must include this command in the configuration file of the *shell* program used by default in the system. The name of this file changes according to which *shell* program it is : bash, sh, zsh, ksh, tcsh, csh... On most Linux systems, which use bash, the file is called « .bashrc » so we should run the following command from our home directory :

```
echo 'export PATH="/home/user/context/texmf-linux-64/bin : "$PATH' >> .bashrc
```

Important note : By executing this step, we will disable the possibility of using other versions of ConTeXt on our system, such as the one incorporated in TeX Live or « ConTeXt Standalone ». If we want to make both versions compatible, it is preferable to use the procedure described in [section 3](#).

2.2 Installing extension modules in LMTX

ConTeXt LMTX does not incorporate a procedure for installing or upgrading the ConTeXt extension modules. However, in ConTeXt wiki there is a *script* that allows you to install and update all the modules along with the rest of the installation.

To do this we need to copy the [aforementioned script](#), paste it into a text file located in the main LMTX installation directory (the one containing `install.sh` or `install.bat`) and run it from a terminal. I have personally verified that this works on a GNU Linux system. I'm not sure if it will work on a Windows system, since I don't have any version of that operating system to check it with.

2.3 Updating LMTX

Updating LMTX is as simple as running the installation program again : it will check the installed files against those on the web server and update as necessary.

If the website from which the files are obtained has changed, we obviously need to also change this address in the installation *script* ; although perhaps it is easier to download a new version of the installation files in the same directory and extract from it the new « `install.sh` » or « `install.bat` » ; or, even easier, unzip the file with the installation program and reinstall without first needing to removing the old files.

2.4 Creating a file that loads the variables into memory needed for LMTX (only GNU/Linux systems)

« ConTeXt Standalone » contains, as we already know, a (« `tex/setup.tex` ») file that loads into memory all the variables needed to run it, but LMTX does not include a similar file. We can, however, easily create it ourselves and store it, for example, as « `setuplmtx` » in the « `tex` » directory. The commands that this file could have would be :

```
export PATH="~/context/LMTX/tex/texmf-linux-64/bin:"$PATH
echo "Adding ~/context/LMTX/tex/texmf-linux-64/bin to PATH"
export TEXROOT="~/context/LMTX/tex"
echo "Setting ~/context/LMTX/tex as TEXROOT"
export OSFONTDIR="~/fonts:/usr/share/fonts:/usr/share/texmf/fonts/opentype/"
echo "Loading font directories into memory"
alias lmtx="~/context/LMTX/tex/texmf-linux-64/bin/context"
echo "Creating an alias to run lmtx"
```

With this, besides loading into memory the paths and variables needed to run LMTX we would be enabling the « `lmtx` » command as a synonym of « `context` ».

After creating this file, before being able to use LMTX, where we intend to use it we should run the following in the terminal :

```
source ~/context/LMTX/tex/setuplmtx
```

all this assuming that LMTX is installed in « `/context/LMTX` » and that we have called this file « `setuplmtx` » and stored it in « `/context/LMTX/tex` ».

The above is what I do, to work with LMTX in the same way I used to work with « ConTEXt Standalone ». However, I do not exclude the possibility that in LMTX it is not necessary, for example, to load into memory the variable OSFONTDIR, since I am struck by the fact that ConTEXt wiki says nothing about this.

3 Using several versions of ConTeXt on the same system (only for Unix-type systems)

The operating system utility called `alias` allows us to associate different names with different versions of ConTeXt. So we can use, for example, the version of ConTeXt included in TeX Live and LMTX ; or the *Standalone* version and LMTX.

For example, if we store the versions of LMTX downloaded in January and August 2020 in different directories, we could write the following two instructions in « `.bashrc` » (or equivalent file read by default when opening a terminal) :

```
alias lmtx-01="/home/user/context/202001/tex/texmf-linux-64/bin/context"
alias lmtx-08="/home/user/context/202008/tex/texmf-linux-64/bin/context"
```

These instructions will associate the names `lmtx-01` with the version of LMTX installed in the « `context/202001` » directory and `lmtx-08` with the version installed in « `context/202008` ».

Annexe B

Commandes pour générer des symboles mathématiques et non mathématiques

Dans les tableaux suivants, vous trouverez les commandes qui génèrent une variété de symboles, vérifiés un par un par moi-même ; la plupart d'entre eux (mais pas tous) sont de préférence destinés à être utilisés en mathématiques.

Je suis conscient que la manière dont elles sont organisées pourrait être améliorée. Le problème est que, étant donné que je suis plutôt issu de la littérature, je ne sais pas à quoi servent beaucoup de ces symboles en mathématiques ; et souvent, je ne suis même pas sûr qu'il s'agisse vraiment de symboles utilisés en mathématiques. C'est pourquoi j'ai fait un groupe des symboles dont je suis raisonnablement sûr qu'ils ne sont pas utilisés en mathématiques, et pour le reste, j'ai regroupé les différents symboles selon certaines formes reconnaissables (triangles, carrés, astérisques, losanges, flèches, points). Pour le reste des symboles, *probablement* ceux utilisés en maths, je les ai classé par ordre alphabétique (de la commande qui les génère).

Monnaie et symboles à usage juridique :

© \copyright	® \registered	¢ \textcent
₱ \textcircledP	¤ \textcurrency	\$ \textdollar
đ \textdong	€ \texteuro	f \textflorin
£ \textsterling	¥ \textyen	™ \trademark

Triangles, cercles, carrés et autres formes :

△ \triangle	○ \bigcirc	□ \square
◀ \triangleleft	◦ \circ	■ \blacksquare
▷ \triangleright	• \bullet	□ \boxdot
▽ \triangledown	◎ \circledast	⊖ \boxminus
▼ \blacktriangledown	◎ \circledcirc	⊕ \boxplus

◀	\blacktriangleleft	⊖	\circledddash	⊗	\boxtimes
▶	\blacktriangleright	⊕	\bigoplus	*	\ast
▲	\blacktriangle	⊗	\bigotimes	✗	\maltese
△	\triangleq	⊕	\oplus	*	\star
◊	\diamond	⊖	\ominus	♣	\clubsuit
◊	\lozenge	⊗	\otimes	♥	\heartsuit
◆	\blacklozenge	∅	\oslash	♠	\spadesuit
◊	\diamondsuit	○	\odot	∅	\varnothing

Flèches :

←	\leftarrow, \gets	→	\rightarrow, \to	↔	\leftrightarrows
↔	\nleftarrow	↔	\nrightarrow	↔	\Leftrightarrow
⟵	\longleftarrow	→	\longrightarrow	⟵	\longleftrightsquigarrow
⇐	\Lleftarrow	⇒	\Rrightarrow	⇐	\Longleftrightsquigarrow
↮	\nLeftarrow	≠	\nrightarrow	↮	\leftrightharpoons
↑	\Lsh	↑	\Rsh	⤵	\leftrightsquigarrow
⤵	\mapsfrom	⤶	\mapsto	⤵	\nLeftrightarrow
⤷	\longmapsfrom	⤸	\longmapsto	⤷	\nleftrightsquigarrow
⤷	\Mapsfrom	⤸	\Mapsto	⤷	\rightleftarrows
⤷	\Longmapsfrom	⤸	\Longmapsto	⤷	\rightleftharpoons
⤷	\leftarrowtail	⤸	\rightarrowtail	⤷	\updownarrow
⤷	\twoheadleftarrow	⤸	\twoheadrightarrow	⤷	\Updownarrow
⤷	\curvearrowleft	⤸	\curvearrowright	⤷	\updownarrows
⤷	\hookleftarrow	⤸	\hookrightarrow	⤷	\uparrow
⤷	\leftharpoondown	⤸	\rightharpoondown	⤷	\upuparrows
⤷	\leftharpoonup	⤸	\rightharpoonup	⤷	\uparrow
⤷	\leftleftarrows	⤸	\rightrightarrows	⤷	\upharpoonleft
⤷	\looparrowleft	⤸	\looparrowright	⤷	\upharpoonright
⤷	\swarrow	⤸	\searrow	⤷	\downarrow
⤷	\nwarrow	⤸	\nearrow	⤷	\Downarrow
⤷	\leftsquigarrow	⤸	\leadssto, \rightsquigarrow	⤷	\downdownarrows
⤷	\iff\$	⤸	\twoheaddownarrow	⤷	\downharpoonleft
⤷	\implies\$	⤸		⤷	\downharpoonright

Ponctuation :

⋮	\because	·	\cdot	\cdot	\cdotp
…	\cdots	·	\centerdot	:	\colon
⋮	\ddots	...	\dots	.	\ldotp
…	\ldots	...	\textellipsis	∴	\therefore
:	\vdots	,	\quotedblbase	"	\quotedbl

Symboles principalement destinés à un usage scientifique :

N	\aleph	II	\amalg	∠	\angle
≈	\approx	≈	\approxeq	≈	\asymp
~	\backsimeq	\backslash	\backslash	⊐	\barwedge
ꝝ	\between	⊓	\bigcap	⊔	\bigcup
□	\bigsqcup	⊕	\biguplus	▽	\bigvee
∧	\bigwedge	⊥	\bot	¤	\bowtie
≈	\Bumpeq	∩	\$\cap\$	¤	\Cap
≈	\circeq	C	\complement	≈	\cong
II	\coprod	U	\cup	ψ	\Cup
≤	\curlyeqprec	≥	\curlyeqsucc	Υ	\curlyvee
Λ	\curlywedge	¬	\$\dashv\$	†	\daggervee
‡	\ddagger, ddag	◊	\diamondsuit	÷	\div
⌘	\divideontimes	÷	\doteq	÷	\doteqdot

+	\dotplus	ℓ	\ell	\emptyset	\emptyset
=	\eqcirc	\gtrless	\gtrless	\eqslantless	\eqslantless
\equiv	\equiv	\eth	\eth	\exists	\exists
$\exists!$	\exists!	\fallingdotseq	\fallingdotseq	\flat	\flat
\forall	\forall	\frown	\frown	\geq	\geq
\gtrapprox	\gtrapprox	\gg	\gg	\ggg	\ggg
\gtrapprox	\gnapprox	\gneqq	\gneqq	\gnsim	\gnsim
\gtreqless	\gtreqless	\gtrdot	\gtrdot	\gtreqless	\gtreqless
\gtreqless	\gtreqless	\gtrless	\gtrless	\gtrsim	\gtrsim
\hbar	\hbar	\heartsuit	\heartsuit	\hslash	\hslash
\intint	\intint	\Im	\Im	\imath	\imath
∞	\$\in\$	∞	\infty	\int	\$\int\$
T	\intercal	\jmath	\jmath	\land	\land
\leftthreetimes	\leftthreetimes	\leq	\leq	\leqq	\leqq
\leqslant	\leqslant	\lessapprox	\lessapprox	\lessdot	\lessdot
\lesseqgtr	\lesseqgtr	\lesseqgtr	\lesseqgtr	\lessgtr	\lessgtr
\lessim	\lessim	\ll	\ll	\lll	\lll
\lnapprox	\lnapprox	\lneq	\lneq	\lneqq	\lneqq
\lnsim	\lnsim	\lor	\lor	\ltimes	\ltimes
\measuredangle	\measuredangle	\models	\models	\mp	\mp
\multimap	\multimap	∇	\nabla	∇	\nabla
\natural	\natural	\cong	\cong	\neq	\neq
$\neg o \lnot$	\neg o \lnot	\nexists	\nexists	\ngeq	\ngeq
\ngtr	\ngtr	\ni	\ni	\nleq	\nleq
\nless	\nless	\nmid	\nmid	$\not\approx$	\not\approx
$\not\equiv$	\not\equiv	$\not\sim$	\not\sim	$\not\simeq$	\not\simeq
\notin	\notin	\parallel	\parallel	\prec	\prec
\nsim	\nsim	\subseteq	\subseteq	\succ	\succ
\nsupseteq	\nsupseteq	\ntriangleleft	\ntriangleleft	\ntrianglelefteq	\ntrianglelefteq
\ntriangleright	\ntriangleright	\ntrianglerighteq	\ntrianglerighteq	\nvDash	\nvDash
\nvDash	\nvDash	\oint	\$\oint\$	\parallel	\parallel
∂	\partial	\perp	\perp	$\text{\scriptsize perthousand}$	\text{\scriptsize perthousand}
\pm	\pm	\prec	\prec	\preccurlyeq	\preccurlyeq
\preceq	\preceq	\precsim	\precsim	\precsim	\precsim
\prime	\prime	\prod	\prod	\propto	\propto
\Re	\Re	\rightthreetimes	\rightthreetimes	\risingdotseq	\risingdotseq
\rtimes	\rtimes	\sharp	\sharp	\sim	\sim
\simeq	\simeq	\smile	\smile	\sphericalangle	\sphericalangle
\sqcap	\sqcap	\sqcup	\sqcup	\sqsubset	\sqsubset
\sqsubseteq	\$\sqsubseteq\$	\sqsupset	\sqsupset	\sqsupseteq	\sqsupseteq
\subset	\subset	\Subset	\Subset	\subseteq	\subseteq
\subsetneq	\subsetneq	\succ	\succ	\succcurlyeq	\succcurlyeq
\succeq	\succeq	\succnsim	\succnsim	\succsim	\succsim
\sum	\sum	\supset	\$\supset\$	\Supset	\Supset
\supseteq	\supseteq	\supsetneq	\supsetneq	\surd	\surd
tpm	\text{tpm}	\times	\times	\top	\top
\triangle	\triangle	\oplus	\oplus	\vdash	\vdash
\Vdash	\Vdash	$\vee o \lor$	\vee o \lor	\veebar	\veebar
\Vert	\$\Vert\$	\Vdash	\Vdash	\wedge	\wedge
\wp	\wp	\wr	\wr		

Autres symboles :

$\text{\textcircled{P}}$	\text{\textcircled{P}}	$\text{\textcircled{S}}$	\text{\textcircled{S}}	$^{\circ}\text{C}$	\celsius
\checkmark	\checkmark	$\text{\textcircled{mho}}$	\text{\textcircled{mho}}	Ω	\ohm
$^{\circ}$	\text{degree}	N	\text{numero}	$\text{\textcircled{v}}$	\text{visible space}

Annexe C

Index des commandes

Les commandes abordées dans cette introduction sont listées dans cet index. Certaines sont simplement mentionnées, presque en passant, auquel cas la page qui apparaît dans l'index indique où elles sont mentionnées. Mais d'autres commandes font l'objet d'une explication plus détaillée. Dans ce cas, seul l'endroit où commence l'explication détaillée est listé dans l'index, bien que la commande puisse être citée à d'autres endroits de l'introduction également.

Ne sont pas inclus dans cet index :

- Les `\stopQuelqueChose` qui ferment une construction précédemment ouverte avec `\startQuelqueChose`, à moins que le texte ne dise quelque chose de spécial à propos de la commande `\stop`, ou qu'il soit traité à un endroit différent de celui où se trouve la commande `\start` correspondante.
- Les commandes visant à générer des symboles, toutes trouvées dans [Appendice B](#).
- Dans le cas des commandes visant à générer un diacritique ou une lettre, et qui ont une version majuscule et une version minuscule, pour générer respectivement la majuscule ou la minuscule, seule la version minuscule est incluse.

a	
<code>\aa</code> 236	<code>\amacron</code> 235
<code>\acute{a}</code> 235	<code>\aring</code> 236
<code>\about</code> 217	<code>\at</code> 217
<code>\breve{a}</code> 235	<code>\atilde</code> 235
<code>\circumflex{a}</code> 235	<code>\atleftmargin</code> 127
<code>\adaptlayout</code> 114	<code>\atpage</code> 220
<code>\adaptpagesize</code> 107	<code>\atrightmargin</code> 127
<code>\adiaeresis</code> 235	
<code>\ae</code> 237	b
<code>\eligature</code> 237	<code>\backslash</code> 55
<code>\grave{a}</code> 235	<code>\\\</code> 281
<code>\alpha</code> 238	<code>\begingroup</code> 76
	<code>\beta</code> 238

\bf	140		c
\bfa	141		\Cap 243
\bfb	141		\c 236
\bfc	141		\ca 255
\bfd	141		\calligraphic 140
\bfx	141		\cap 243
\bfxx	141		\ccedilla 236
\bgroup	75, 76		\cg 140
\bi	140		\chapter 162
\bia	141		\chi 238
\bib	141		\cite 230
\bic	141		\clip 355
\bid	141		\clubpenalties 291
\bigbodyfont	143		\color 148, 149
\bix	141		\colored 149
\bixx	141		\completecontent 188
\blackrule	328		\completelistofchemicals 204
\blackrules	328		\completelistoffigures 204
\blank	87, 273		\completelistofgraphics 204
\bold	140		\completelistofintermezzi 204
\bolditalic	140		\completelistoftables 358
\boldslanted	140		\completetelist 202
\break	281		\completindex 208
\bs	140		\completelistoffigures 350
\bsa	141		\completelistoftables 204
\bsb	141		\crlf 281
\bsc	141		\currentdate 261
\bsd	141		
\bsx	141		
\bsxx	141		
\bTABLE	365		d
\bTABLEbody	365		\date 261
\bTABLEfoot	365		\de 256
\bTABLEhead	365		\define 71
\bTABLEnext	365		\definealternativestyle 145
\bTD	365		\defineblank 274
\bTH	365		\definebodyfontenvironment 136,
\bTR	365		142
\buildmathaccent	239		\definebodyfontswitch 144
\buildtextaccent	239		\definecapitals 244
\buildtextbootomcomma	239		\definecharacter 238
\buildtextbottomdot	239		\definecharacterkerning 251
\buildtextcedilla	239		\definecolor 152
\buildtextgrave	239		\definecombination 380
\buildtextmacron	239		\definecombinedlist 204
\buildtexttognek	239		\defineconversion 175
			\defineconversionset 118
			\definedelimitedtext 263
			\definedescription 320

\ 155
\defineenumeration 323
\definefloat 375, 381
\definefontfeature 159, 237
\definefontstyle 144
\definefontsynonym 156
\defineframed 332
\defineframedtext 332
\definehead 182
\definehighlight 148
\defineinterlinespace 284, 285
\defineitemgroup 319
\defineitems 319
\definelayout 114
\definelinenumbering 288
\definelines 286
\definelist 201
\definemargindata 130
\definenarrower 269
\definenote 299
\definepapersize 106, 107
\defineparagraphs 308
\defineregister 209
\defineresetset 170
\definesectionblock 184
\definestartstop 74
\definestretched 251
\definestructureconversionset 173
\definestructureseparatorset 174
\definesymbol 312
\definetabulate 365
\definetext 124
\definetype 247
\definotyping 247
\delta 238
\dontleavehmode 140

e
\acute{e} 235
\breve{e} 235
\circumflex{e} 235
\ediaeresis 235
\grave{e} 235
\egroup 75, 76
\em 146
\macron{e} 235
\mdash 89
\en 255

\enableregime 84
\endash 89
\endgraf 266
\endgroup 76
\endnote 296
\enskip 251
\environnement 96
\epsilon 238
\es 255
\eta 238
\tilde{e} 235
\externalfigure 345, 352

f
\fillinline 327
\fillrules 328
\footnote 296
\footnotetext 298
\fr 255
\framed 331
\from 226

g
\gamma 238
\getbuffer 338
\getmarking 123
\godown 274
\goto 227

h
\H 236
\HL 362
\hairline 326
\handwritten 140
\hbox 279
\head 313
\hfill 251
\high 246
\hl 327
\hskip 251, 252
\hw 140
\hyphen 89
\hyphenatedurl 225
\hyphenatedurlseparator 226
\hyphenation 279

i
\i 236

\iacute 235
 \ibreve 235
 \icircumflex 235
 \idiaeresis 235
 \igrave 235
 \imacron 235
 \in 217
 \index 206
 \inframed 331
 \ininner 127
 \ininneredge 127
 \ininnermargin 127
 \inleft 127
 \inleftedge 127
 \inleftmargin 127
 \inmargin 127
 \inother 127
 \inouter 127
 \inouteredge 127
 \inoutermargin 127
 \input 93
 \inright 127
 \inrightedge 127
 \inrightmargin 127
 \iota 238
 \it 140
 \ita 141
 \italic 140
 \italicbold 140
 \itb 141
 \ite 141
 \itd 141
 \item 313
 \items 318
 \itilde 235
 \its 314
 \itx 141
 \itxx 141

j
 \j 236

k
 \kappa 238
 \kcedilla 236

l
 \l 236

\labeltext 260
 \lambda 238
 \language 255
 \lastpagenumber 119
 \lastrealpagenumber 119
 \lastuserpagenumber 119
 \lcedilla 236
 \leftaligned 290
 \letterbackslash 226
 \letterescape 226
 \letterhash 226
 \letterhat 55
 \letterpercent 226
 \lettertilde 55
 \linenote 296
 \loadinstalledlanguages 257
 \lohi 246
 \low 246

m
 \mainlanguage 255, 279
 \mar 314
 \margintext 127
 \mediaeval 140
 \midaligned 290
 \minus 89
 \mirror 355
 \mono 140
 \month 261
 \mu 238

n
 \NB 364
 \NC 363
 \NN 364
 \NR 363
 \namedstructurevariable 165
 \ncedilla 236
 \noindentation 268
 \nolist 165, 168
 \nomarking 165, 168
 \nop 313
 \normal 140
 \note 297, 298
 \notesenabledfalse 305
 \notesenabledtrue 305
 \nowhitespace 272
 \nu 238

o	\project 99 \projet 100 \psi 238
\o 236	
\oacute 235	
\obreve 235	
\ocircumflex 235	
\odiaeresis 235	
\oe 237	
\oeligature 237	
\ograve 235	
\omacron 235	
\omega 238	
\omicron 238	
\os 140	
\otilde 235	
\overbar 330	
\overbars 330	
\overstrike 330	
\overstrikes 330	
 p	
\page 120	
\pagename 119, 123	
\pageref 215	
\par 266	
\parskip 79	
\part 162	
\phi 238	
\pi 238	
\placebookmarks 228	
\placechemical 375	
\placecontent 188	
\placefigure 347	
\placefloat 375	
\placegraphic 375	
\placeindex 208	
\placeintermezzo 375	
\placelist 202	
\placelistofchemicals 204	
\placelistoffigures 204, 350	
\placelistofintermezzi 204	
\placelistofpublications 230	
\placelistofgraphics 204	
\placelistoftables 204, 358	
\placelocalfootnotes 298	
\placenotes 298	
\placetable 357	
\pretolerance 279	
\product 98	
 q	
\qqquad 251	
\quad 251	
\quotation 263	
\quote 263	
 r	
\ReadFile 94	
\r 236	
\ran 314	
\rcedilla 236	
\readfile 94	
\realpagenumber 119	
\ref 218	
\reference 215	
\regular 140	
reserved characters	
\{ 55	
\} 55	
\\$ 55	
\backslash 55	
\letterhat 55	
\lettertilde 55	
\# 55	
\% 55	
\& 55	
_ 55	
\ 55	
\rho 238	
\rightaligned 290	
\rm 140	
\rma 141	
\rmb 141	
\rmc 141	
\rmd 141	
\rmx 141	
\rmxx 141	
\roman 140	
\rotate 355	
 s	
\sans 140	
\sansserif 140	
\sc 140	

```

\scedilla 236
\section 162
\seeindex 208
\serif 140
\setdefaultpenalties 292
\sethyphenatedurlafter 225
\sethyphenatedurlbefore 225
\sethyphenatedurlnormal 225
\setupalign 289
\setuparranging 105, 113
\setupbackgrounds 149
\setupblackrules 328
\setupblank 274
\setupbodyfont 135, 139
\setupbottomtexts 127
\setupcapitals 244
\setupcaption 378
\setupcaptions 378
\setupcharacterkerning 251
\setupcharacterspacing 258
\setupcolors 149
\setupcolumns 307
\setupcombinedlist 189
\setupendnotes 301
\setupenumeration 325
\setupexternalfigures 101, 346, 352
\setupfillinlines 327
\setupfloat 380
\setupfloats 380
\setupfooter 123
\ 124
\setupfootertexts 122, 125
\setupfootnotes 301
\setupframed 332
\setupframedtext 332
\setuphead 166
\setupheader 123
\ 124
\setupheadertexts 122, 125
\setupheadnumber 169
\setupheads 166
\setupheadtext 188
\setuphyphenmark 253
\setupindenting 267
\setupinteraction 222
\setupinterlinespace 283
\setuplabeltext 259, 260, 377
\setuplanguage 257
\setuplayout 109, 112
\setuplinenote 296
\setuplinenumbering 287
\setuplines 286
\setuplist 193, 202
\setupmargindata 129
\setupnarrower 269
\setupnotation 301
\setupnotations 301
\setupnote 301
\setupnotes 301
\setuppagenumbering 117
\setuppapersize 104
\setupparagraphs 308
\setupregister 209
\setupsectionblock 183
\setupspacing 248
\setupstretched 251
\setupTABLE 365
\setuptable 368
\setuptables 358
\setuptabulate 365
\setupextrule 330
\setupthinrules 327
\setuptolerance 280, 291
\setupoptexts 127
\setupotype 247
\setuptyping 247
\setupurl 225
\setupuserpagenumber 117
\setupwhitespace 271
\showbodyfontenvironment 143
\showbtxdatasetfields 230
\showcolor 151
\showcolorcomponents 151
\showfont 137
\showframe 111
\showinstalledlanguages 257
\showlayouts 111
\showsetups 111
\showsymbolset 242
\sigma 238
\sl 140
\sla 141
\slanted 140
\slantedbold 140
\slb 141
\slc 141

```

\sld 141
 \slx 141
 \slxx 141
 \smalcaps 140
 \smallbodyfont 143
 \smallbold 143
 \smallbolditalic 143
 \smallboldslanted 143
 \smallitalicbold 143
 \smallslanted 143
 \smallslantedbold 143
 \somewhere 219
 \space 251
 \ss 140, 236
 \ssa 141
 \ssb 141
 \ssc 141
 \ssd 141
 \ssx 141
 \ssxx 141
 \start 76
 \startalignment 290
 \startappendices 183
 \startbackmatter 183
 \startbodymatter 183
 \startbuffer 338
 \startchapter 162
 \startchemical 338
 \startcolor 151
 \startcolumns 306
 \startcombination 379
 \startcombinations 338
 \startcomponent 98
 \startenvironment 96
 \startfiguretext 351
 \startformula 339
 \startframedtext 331
 \startfrontmatter 183
 \starthiding 339
 \startitem 313
 \startitemize 312
 \startlegend 339
 \startlinecorrection 340
 \startlinenumbering 287, 288
 \startlines 286
 \startlocalfootnotes 298
 \startMPpage 107
 \startmode 340
 \startnarrower 268, 271, 272
 \startnotmode 340
 \startopposite 340
 \startpacked 272, 274
 \startpagefigure 107
 \startpart 162
 \startproduct 98
 \startproject 99
 \startquotation 340
 \startsection 162
 \startsectionblockenvironment 184
 \startsetups 74, 292
 \startstandardmakeup 340
 \startsubject 162
 \startsubsection 162
 \startsubsubsection 162
 \startsubsubsubsection 162
 \startsubsubsubsubsection 162
 \startsubsubsubsubsubsection 162
 \startTEXpage 107
 \starttabulate 358
 \starttext 91
 \starttexrule 329
 \starttitle 162
 \starttypescript 156
 \starttyping 246
 \stop 76
 \stoptext 91
 \stretched 249
 \sub 314
 \subject 162
 \subsection 162
 \subsubject 162
 \subsubsection 162
 \subsubsubsection 162
 \subsubsubsubsection 162
 \subsubsubsubsubsection 162
 \support 140
 \switchtobodyfont 139
 \sym 314
 \symbol 242

t

\TB 364
 \TeX 24, 25
 \tau 238
 \tcedilla 236
 \teletype 140

```

\tex  247
\textheight 113
\textreference 215
\textrule 329
\textwidth 113
\tf 140
\ta 141
\fb 141
\fc 141
\fd 141
\fx 141
\txx 141
\theta 238
\thinrule 326, 327
\thinrules 327
\title 162
\tolerance 279
\translate 262
\tt 140
\ta 141
\tb 141
\tc 141
\td 141
\tx 141
\txx 141
\tx 141
\txx 141
\typ 247
\type 246
\typebuffer 338

u
\u 235
\acute{u} 235
\breve{u} 235
\circumflex{u} 235
\diaeresis{u} 235
\grave{u} 235
\macron{u} 235
\bar{u} 330
\bars{u} 330
\unhyphenated{u} 279
\upsilon 238

\usebtxdataset 230
\usecolors 150
\useexternalfigure 345
\usemodule 257
\usepath 101
\useregime 84
\userpagenumber 119
\usesymbols 242
\useURL 224
\useexternalfigure 352
\utilde 235

v
\VL 362
\varepsilon 238
\varkappa 238
\varphi 238
\varpi 238
\varrho 238
\varsigma 238
\vartheta 238
\vbox 121
\vfill 275
\vl 327
\vskip 274

w
\WORD 243
\WORDS 243
\Word 243
\Words 243
\whitespace 272
\widowpenalties 291
\word 243
\wordright 252, 290
\writebetweenlist 199
\writetolist 198

x
\xi 238

z
\zeta 238

```

Annexe D

Index

a

aide et ressources [33](#)

c

compilation [41](#)
composition [20](#)

d

define [71](#)
definestartstop [74](#)

e

encode [82](#)
espace [85](#)
espace vide [85](#)

f

fichier
composant [98](#)
environnement [96](#)
produit [98](#)
projet [99](#)
fichiers
chemin [101](#)
fonts [154](#)
OSFONDIR [154](#)

l

langages de balisage
balises [22](#)
markup [22](#)

m

Mark II [18](#)
Mark IV [18](#)
macro-structure [118](#)
moteurs \TeX [24](#)
mtxrun [154](#)

p

page
dimension [104](#)
préambule [44](#)

r

réécriture [20](#)
règles syntaxiques
commande [68](#)
repertoire
chemin [101](#)

s

saut de ligne [87](#)
setupwhitespace [79](#)
structure
chemin [101](#)
composant [98](#)
produit [98](#)
projet [99](#)

t

tabulation [85](#)
tirets [88](#)
trait d'union [88](#)

Une courte (?) introduction à ConTeXt Mark IV