

Une courte (?) introduction à ConT_EXt Mark IV

Une courte (?) introduction à ConTeXt Mark IV

Version 1.6 [29 mai 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 indiquée, 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](#).
- 29 mai 2021: Version 1.6 : Corrections suggérées après une nouvelle lecture du document, à l’occasion de sa traduction en anglais

Table des matières

Préface	5
I Qu'est ce que ConT_EXt ? Comment travailler avec ?	13
1 ConT_EXt : une vue d'ensemble	15
1.1 Qu'est-ce que ConT _E Xt ?	15
1.2 La composition typographique de document	16
1.3 Les langages de balisage	17
1.4 T _E X et ses dérivés	19
1.5 ConT _E Xt	21
2 Notre premier fichier source	31
2.1 Préparation de l'expérience outils nécessaires	31
2.2 L'expérience elle-même	33
2.3 La structure de notre fichier d'exemple	38
2.4 Quelques détails supplémentaires sur la façon d'exécuter « context » ..	39
2.5 Traitement des erreurs	40
3 Les commandes et autres concepts fondamentaux de ConT_EXt .	45
3.1 Les caractères réservés de ConT _E Xt	46
3.2 Les commandes à proprement parler	49
3.3 Périmètre des commandes	52
3.4 Options de fonctionnement des commandes	55
3.5 Résumé sur la syntaxe des commandes et des options, et sur l'utilisation des crochets et des accolades lors de leur appel.	58
3.6 La liste officielle des commandes ConT _E Xt	60
3.7 Définir de nouvelles commandes	61
3.8 Autres concepts fondamentaux	65
3.9 Méthode d'auto apprentissage pour ConT _E Xt	69
4 Fichiers sources et projets	73
4.1 Codage des fichiers sources	73
4.2 Caractères dans le(s) fichier(s) source(s) que ConT _E Xt traite d'une manière spéciale	75
4.3 Projet simple et projet multi-fichiers	80
4.4 Structure du fichier source d'un projet simple	81
4.5 Gestion multi-fichiers à la T _E X	82
4.6 Gestion multi-fichiers à la ConT _E Xt	85

II	Global aspects of the document	91
5	Pages et pagination	93
5.1	Taille de la page	93
5.2	Éléments sur la page	97
5.3	Mise en page (<code>\setuplayout</code>)	100
5.4	Numérotation des pages	104
5.5	Sauts de page forcés ou suggérés	107
5.6	En-têtes et pieds de page	109
5.7	Insertion d'éléments de texte dans les bords de page et les marges	114
6	Polices d'écriture et couleurs dans ConTeXt	117
6.1	Polices de caractères incluses dans « ConTeXt Standalone »	117
6.2	Caractéristiques d'une police	118
6.3	Définition de la police principale du document	121
6.4	Modification de la police ou de certaines de ses caractéristiques	125
6.5	Autres questions relatives à l'utilisation de styles alternatifs	132
6.6	Utilisation et configuration des couleurs	134
6.7	Bonus 1 - Utilisation des polices du système d'exploitation	140
7	Structure du document	147
7.1	Les divisions structurelles d'un document	147
7.2	Types et hiérarchie des sections	148
7.3	Syntaxe commune des commandes liées aux sections	150
7.4	Format et configuration des sections et de leurs titres	152
7.5	Définir de nouvelles commandes de section	167
7.6	La macrostructure du document	168
8	Table of contents, indexes, lists	171
8.1	Table of contents	171
8.2	Lists, combined lists and table of contents based on a list	181
8.3	Index	184
III	Particular issues	189
	Appendices	191
A	Index	193

Préface*

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 – 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.

* 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.

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 ?? explique comment installer l'une ou l'autre des distributions.

³ Pour la liste, voir section ??.

Documentation

Il existe une documentation considérable sur ConT_EXt, presque exclusivement en anglais. Par exemple, ce que l'on considère comme étant la distribution *officielle* de ConT_EXt – appelée « ConT_EXt 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 ConT_EXt) 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 « ConT_EXt Standalone » mais également quelques autres. Cependant, cette énorme documentation n'est pas particulièrement utile durant la phase d'apprentissage de ConT_EXt, 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 « ConT_EXt Standalone » appelle « manuels », un seul suppose que le lecteur ne connaît rien sur ConT_EXt. Il s'agit du document intitulé « [ConT_EXt Mark IV, an Excursion](#) » ou en français « ConT_EXt 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 ConT_EXt. 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 « [ConT_EXt 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 titanesque), et d'autre part il était (est) destiné à la version précédente de ConT_EXt (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 ConT_EXt Mark II et ConT_EXt 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 « ConT_EXt Standalone ». Pourtant, malgré ces défauts, et une fois lu « [ConT_EXt Mark IV, an Excursion](#) », le manuel reste le meilleur document pour continuer à apprendre ConT_EXt. Autres informations également toujours très utiles pour démarrer avec ConT_EXt, celles contenues le sur [ConT_EXt Garden wiki](#), qui, au moment où nous écrivons ces lignes, est plein remaniement et présente une structure beaucoup plus claire, bien qu'elles mélangent 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 « [ConT_EXt 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 : « [ConT_EXt Mark IV, an Excursion](#) », « [ConT_EXt Reference Manual 2013](#) », le contenu de [ConT_EXt Garden wiki](#), et la liste officielle des commandes « [ConT_EXt 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 ConT_EXt Mark IV » ou « Ce que j'ai appris sur ConT_EXt 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 ConT_EXt. Il est certain, malgré que la documentation ait selon moi certaines lacunes, que ConT_EXt est un outil vraiment utile et polyvalent, pour lequel l'effort d'apprentissage vaut sans aucun doute la peine. **Grâce à ConT_EXt, 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 resurgissent de temps en temps dans ce document. Je ne veux pas qu'il y ait de malentendu : je suis immensément reconnaissant aux créateurs de ConT_EXt 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 ConT_EXt, 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 différencient les deux versions de ConT_EXt, 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 ConT_EXt pour 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.

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

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.



de commandes et d'options de ConT_EXtest 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 T_EX et ConT_EXt, bien que j'espère qu'elle pourra également être utile à ceux qui s'approchent pour la première fois de T_EX or L^AT_EX (le plus populaire des outils dérivés de T_EX). 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 ConT_EXt, le nouveau venu dans cet écosystème fascinant.

Être un néophyte ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt ou à T_EX implique que j'ai dû y inclure des informations concernant non pas à ConT_EXt mais à T_EX, 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 ConT_EXt ou bien de T_EX. 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 T_EX.

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 ConT_EXt, 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 ConT_EXt ainsi que certaines questions relatives aux fichiers sources de ConT_EXt

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 T_EX ou L^AT_EX, 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 ConT_EXt ([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^AT_EX et ConT_EXt. 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^AT_EX comprendra immédiatement les principales différences dans la syntaxe des deux langages, ainsi que la façon dont ConT_EXt permet de configurer et de personnaliser le fonctionnement de presque toutes ses commandes.
- Les informations relatives à la gestion des projets ConT_EXt multi-fichiers ([Chapitre 4](#)), qui se distingue de celles des autres systèmes basés sur T_EX.
- **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 ?? 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 ConTeXt pour la gestion de ces références qui permet des *sauts* entre différentes 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 ?? 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 ?? et ?? 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 ?? 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 ?? 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 ConT_EXt se termine par trois [Annexes](#). L'une concerne l'installation de ConT_EXt 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 ConT_EXt 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 ConT_EXt et ne prétend pas le faire, j'ai intitulé ce document « une courte (?) introduction à ConT_EXt 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 ConT_EXt ; 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 *envisager 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, HYNÄ ET SCHLEGL, en anglais « *The Not So Short Introduction to L^AT_EX 2_ε* » et en français « *Une courte (?) introduction à L^AT_EX 2_ε* ». 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 ConT_EXt. 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 ConT_EXt, 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
Été 2020

I

Qu'est ce que ConTExT ?

Comment travailler avec ?

Chapitre 1

ConT_EXt : une vue d'ensemble

Table of Contents: 1.1 Qu'est-ce que ConT_EXt ?; 1.2 La composition typographique de document; 1.3 Les langages de balisage; 1.4 T_EX et ses dérivés; 1.4.1 Les moteurs T_EX; 1.4.2 Les formats dérivés de T_EX; 1.5 ConT_EXt; 1.5.1 Une brève histoire de ConT_EXt; LMTX et autres implémentations alternatives de Mark IV 24 1.5.2 ConT_EXt versus L^AT_EX; 1.5.3 La logique de travail avec ConT_EXt; 1.5.4 Obtenir de l'aide sur ConT_EXt;

1.1 Qu'est-ce que ConT_EXt ?

ConT_EXt 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 ConT_EXt.

- Il existe deux versions de ConT_EXt appelées respectivement Mark II et Mark IV. ConT_EXt 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. ConT_EXt 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 ConT_EXt Mark IV.
- ConT_EXt est un logiciel libre. Le programme lui-même (c'est-à-dire l'ensemble des outils logiciels qui composent ConT_EXt) 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.
- ConT_EXt 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 ConT_EXt :

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

- 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 rédigeons 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 réserver 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., on 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 clarté, 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 ConT_EXt 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

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.

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 T_EX 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 T_EX et ses dérivés

T_EX 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 T_EX, 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 «glyphes»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 T_EX, 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 T_EX 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 traitée par T_EX est d'environ 0,000005356 millimètre.

Le nom T_EX vient de la racine du mot grec τεχνη, 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,

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).

avec le « E » central légèrement décalé vers le bas pour faciliter un rapprochement entre le « T » et le « X » ; c'est-à-dire : « T_EX » . 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 ».

Cette convention est suivie dans tous les dérivés de T_EX qui l'incluent dans leur propre nom, comme par exemple ConT_EXt, qui lorsqu'il est écrit en mode texte doit être écrit « ConTeXt ».

1.4.1 Les moteurs T_EX

Le programme T_EX 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 « T_EX ». 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 T_EX* (engine en anglais). En dehors du programme original, les principaux moteurs T_EX sont, par ordre chronologique d'apparition pdfT_EX, ϵ -T_EX, X_YT_EX et LuaT_EX. Chacun d'entre eux est censé intégrer les améliorations de son prédécesseur. Ces améliorations, en revanche, jusqu'à l'apparition de LuaT_EX, 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 T_EX à utiliser fait l'objet d'un débat animé dans l'univers T_EX. Je ne m'y attarderai pas ici, car ConT_EXt Mark IV ne fonctionne qu'avec LuaT_EX. En fait, dans le monde de ConT_EXt la discussion sur les moteurs devient une discussion sur l'utilisation de Mark II (qui fonctionne avec pdfT_EXet X_YT_EX) ou Mark IV (qui fonctionne avec LuaT_EX).

1.4.2 Les formats dérivés de T_EX

Le noyau, ou cœur, de T_EX 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 T_EX 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 T_EX qui imprime votre logo (`\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 « `\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 \TeX Book* »), 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 \LaTeX 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 \LaTeX 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

ConT_EXt pour 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*), ConT_EXt propose 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 K_NU_TH a co-développé avec T_EX.
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 T_EX, MetaPost, LUA et XML, permettant d'écrire et de traiter des documents dans n'importe lequel de ces langages, ou qui mélange 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 ConT_EXt de « technologie hybride » dédié à la composition typographique de documents. C'est également ce qui fait de ConT_EXt un système extraordinairement avancé et puissant.

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

ConT_EXt en revanche est beaucoup plus moderne que T_EX. Lorsque T_EX 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, ConT_EXt 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 ConT_EXt

ConT_EXt 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 T_EX 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 ConT_EXt 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 « ConT_EXt » a été utilisé pour s'y référer. Ce nom est censé signifier « Texte avec T_EX » (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 « T_EX », « texte » et « contexte ».

Par conséquent, bien que ConT_EXt soit dérivé de T_EX (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 ConT_EXt 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 documentatino « officielle » de ConT_EXt 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, ConT_EXt Mark II fonctionnait avec le *moteur* T_EX pdfT_EX. Plus tard, lorsque le nouveau moteur X_YT_EX est apparu, ConT_EXt Mark II a été modifié pour en permettre l'utilisation, qui présentait de nombreux avantages par rapport à pdfT_EX. Des années plus tard encore, lorsque le moteur LuaT_EXa été développé, il a été décidé de reconfigurer le fonctionnement interne de ConT_EXt Mark II pour intégrer toutes les nouvelles possibilités offertes par ce dernier moteur. C'est ainsi qu'est né ConT_EXt Mark IV, qui a été présenté en 2007, immédiatement après la présentation de LuaT_EX. La décision d'adapter ConT_EXt à LuaT_EXa très probablement été influencée par le fait que deux des trois principaux développeurs de ConT_EXt, HANS HAGEN et TACO HOEKWATER, font également partie de l'équipe de développement de LuaT_EX. Par conséquent, ConT_EXt Mark IV et LuaT_EX sont nés simultanément et ont été développés à l'unisson. Il existe une synergie entre ConT_EXt et LuaT_EX et qui n'existe avec aucun autre dérivé de T_EX ; et je ne pense pas qu'aucun d'entre eux ne profite des possibilités de LuaT_EX comme ConT_EXt 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 ConT_EXt, 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 utilisateur, 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 ConT_EXt. 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 ConT_EXt, 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 ConT_EXt 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 ConT_EXt que vous installez à partir de leur wiki, « ConT_EXt 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 « ConT_EXt 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 « ConT_EXt Standalone »).

LMTX et autres implémentations alternatives de Mark IV

Les développeurs de ConT_EXt 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 T_EX, 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 luametaT_EX : un nouveau *moteur* de T_EX qui est une version simplifiée et optimisée de LuaT_EX, développé en vue d'économiser les ressources informatiques et d'offrir une solution T_EX aussi minimaliste 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 LuaT_EX, ou compiler avec luametaT_EX. Une procédure pour attribuer un nom de commande différent à chacune des installations (section ??) est expliquée dans l'annexe ??, relative à l'installation de ConTeXt.

1.5.2 ConTeXt versus L^AT_EX

Comme le format dérivé de T_EX 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 T_EX ; 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.

L^AT_EX vise à faciliter l'utilisation de T_EX, 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 L^AT_EX. En d'autres termes, la simplification de l'utilisation de T_EX est obtenue au prix d'une limitation de son immense flexibilité, par la prédéfinition 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 T_EX que L^AT_EX.

Cette différence de philosophie et d'objectifs fondamentaux se traduit à son tour par une différence de mise en œuvre. Parce que L^AT_EX, qui tend à simplifier au maximum, n'a pas besoin d'utiliser toutes les ressources de T_EX. 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 L^AT_EX est à la fois une force et une faiblesse : une force, car l'énorme popularité de L^AT_EX, 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 L^AT_EX 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 L^AT_EX et son extensibilité par le biais de paquets, ConT_EXt est conçu pour intégrer et rendre accessibles toutes — ou presque toutes — les possibilités typographiques de T_EX, de sorte que sa conception est beaucoup plus monolithique, mais, en même temps, il est aussi plus modulaire : le noyau ConT_EXt 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 commande.

Il est vrai que ConT_EXt propose des *modules* d'extension dont on pourrait considérer qu'ils ont une fonction similaire à celle des paquets de L^AT_EX, mais la vérité est que la fonction des deux est très différente : les modules de ConT_EXt 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 ConT_EXt 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 L^AT_EX ; 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 ConT_EXt, 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 ConT_EXt

Lorsque nous travaillons avec ConT_EXt, 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 ConT_EXt) 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 ConT_EXt, 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 illibbles 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 ConT_EXt, 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt

Tant que nous sommes des débutants, le meilleur endroit pour trouver de l'aide sur ConT_EXt 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 ConT_EXt nous jouera un sale tour car une recherche d'informations sur « contexte » renverrait des millions de résultats et la plupart d'entre eux n'auraient aucun rapport avec ce que nous recherchons. Pour rechercher des informations sur ConT_EXt, vous devez ajouter quelque chose au nom « context » ; par exemple, « tex », « luatex », « Mark IV » , « Hans Hagen » (un des créateurs de ConT_EXt), « 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 ConT_EXt, et si l'on maîtrise bien l'anglais, on peut consulter l'un des nombreux documents inclus dans « ConT_EXt Standalone » ou demander de l'aide :

- soit sur [TeX – LaTeX Stack Exchange](#) et en particulier les questions taguées « ConT_EXt »

- soit sur la liste de diffusion propre à ConT_EXt [NTG-context](#) et son [moteur de recherche](#).

Cette dernière liste diffusion implique les personnes les plus compétents sur ConT_EXt, 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 devons 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 T_EX, 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 T_EX 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 tapés 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

« ConT_EXt Standalone » dont les instructions d’installation se trouvent sur le [wiki de ConT_EXt](#). 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 visualisateurs 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 ñes) ou de ne pas permettre de sélectionner le texte, d’envoyer le document à l’imprimante ; c’est juste un visualisateur, 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 ConT_EXt... 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 ConT_EXt 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 ConT_EXt que nous avons installée est « ConT_EXt Standalone », nous devons tout d'abord exécuter le *script* qui indique au terminal les chemins et l'emplacement des fichiers dont ConT_EXt 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 ConT_EXt 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 « ConT_EXt 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 avons 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 ConT_EXt. 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 pourrions 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt, 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 ConT_EXt 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 « ConT_EXt 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 ConT_EXt qui a été interrompue par l'erreur.

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

La [figure B.1](#) montre le contenu de « la-maison-sur-le-port.pdf ». Nous pouvons voir que ConT_EXt 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 huitième ligne. ConT_EXt, 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 ConT_EXt 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 : ConT_EXt a transformé le fichier source et a généré un fichier dans lequel nous avons un document formaté selon les instructions qui étaient incluses dans

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

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 ConT_EXt.

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 ConT_EXt 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 ConT_EXt 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 ConT_EXt, 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 ConT_EXt, 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 ConT_EXt fait la différence entre les majuscules et les minuscules et considère donc que « TeX » et « Tex » sont des mots différents ; ou si les options utilisées 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 ConT_EXt 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 ConT_EXt 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 `\startttext` nous avons 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 compilons 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 ConT_EXt. 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 « `startttext` » (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 devons continuer à enquêter jusqu'à ce que nous découvriions les raisons de l'erreur et sa possible solution.

Dans la pratique, lorsqu'on crée un document relativement volumineux avec Con-TeXt, 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 ConT_EXt

Table of Contents: 3.1 Les caractères réservés de ConT_EXt; 3.2 Les commandes à proprement parler; 3.3 Périmètre des commandes; 3.3.1 Les commandes qui nécessitent 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 ConT_EXt; 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 ConT_EXt;

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 ConT_EXt 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 ConT_EXt (en fait, du fonctionnement de T_EX), 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 de suite. La plupart des instructions de ConT_EXt 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 ConT_EXt parle généralement d'une *commande* lorsqu'elle adopte le point de vue de l'utilisateur, et d'une *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.

Les *commandes* sont des ordres donnés au programme ConT_EXt pour qu'il fasse quelque chose. Nous *contrôlons* les performances du programme par leur intermédiaire. Ainsi K_NU_TH, le père de T_EX, 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'utiliserai lorsque je penserai qu'il est important de distinguer entre *symboles de contrôle* et *mots de contrôle*.

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

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.

3.1 Les caractères réservés de ConT_EXt

Lorsque ConT_EXt 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 ConT_EXt sont ce qui lui permet de faire cette distinction. En principe, ConT_EXt 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. ConT_EXt 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 :

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

ConT_EXt 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 ConT_EXt 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 rappor 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 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 ??.
- | 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`.
- ^ 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
\	<code>\backslash</code>
%	<code>\%</code>
{	<code>\{</code>
}	<code>\}</code>
#	<code>\#</code>
~	<code>\lettertilde</code>
	<code>\ </code>
\$	<code>\\$</code>
_	<code>_</code>
^	<code>\letterhat</code>
&	<code>\&</code>

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 considèrera 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 ConT_EXt. 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 ($_ \wedge \&$) : 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 ConT_EXt 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

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 :

- a. **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 pourrions 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 T_EXBook », 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 ».

- b. **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 ConT_EXt 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 ConT_EXt ne contient aucune règle sur les noms de commande, tout comme le reste des « manuels » inclus avec « ConT_EXt Standalone ». Ce que j'ai



⁹ **Note:** par convention, pour illustrer quelque chose dans cette introduction, les exemple de code source utilise une police à espacement fixe. une coloration syntaxique cohérente de ConTeXt dans un cadre de fond gris. Le résultat de la compilation est présenté dans un cadre de fond de couleur jaune foncé.

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 49](#)) 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 $\text{\texttt{\textbackslash txt-}}$, Con\TeX t la remplace par $\text{\texttt{\textbackslash TeX}}$ 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 $\text{\texttt{\textbackslash define}}$, qui est une commande spécifiquement Con\TeX t pour définir des macros.

3.3 Périmètre des commandes

3.3.1 Les commandes qui nécessitent ou pas un périmètre d'application

De nombreuses commandes Con\TeX t 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 $\text{\texttt{\textbackslash bf}}$ active le gras, et elle restera active jusqu'à ce qu'elle trouve une commande *incompatible* comme, par exemple, $\text{\texttt{\textbackslash tf}}$, ou $\text{\texttt{\textbackslash it}}$.

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 {\textit The \TeX Book}, {\textsc Knuth} explained \bf{everything} you need to know about \TeX.
```

In *The \TeX Book*, K $\text{\textsc{NUTH}}$ explained **everything you need to know about \TeX** .

crée deux groupes, l'un pour déterminer la portée de la commande $\text{\texttt{\textbackslash it}}$ (italique) et l'autre pour déterminer la portée de la commande $\text{\texttt{\textbackslash sc}}$ (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 $\text{\texttt{\textbackslash framed}}$

: 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'appliquent, 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 appelées *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, ConT_EXt 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 ConT_EXt 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 prédéfinis, 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 prédéfinis : 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'incluions 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 distrinctes

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 :

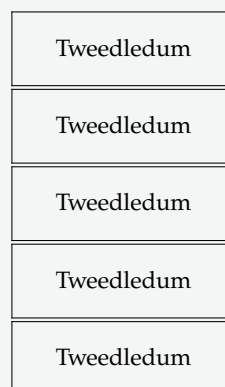
```
\framed[width=3cm,height=1cm]{Tweedledum}

\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 ConT_EXt 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 *prédéfinir* 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 créons. 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 ConT_EXt

- 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 omises : 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 ??) 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 section
NERA PAS

2 mon titre[5] avec crochets

Du texte pour cette section FONCTIONNERA

3.6 La liste officielle des commandes ConT_EXt

Parmi la documentation de ConT_EXt 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 ConT_EXt. 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 ConT_EXt : 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 ConT_EXt, car c'est là que nous pouvons savoir si une certaine commande existe ou non ; ceci est particulièrement utile, compte tenu de la combinaison COMMANDE (OU ENVIRONNEMENT) + `setup-COMMANDE` + `defineCOMMANDE`. 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 ConT_EXt. 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 ConT_EXt. 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 ConT_EXt 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 répercutée 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=5pt,
frame=off,strut=no]{%debutZ
\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 ConT_EXt.

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 substituer 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
```

Anne Smith
Consultant

Marseille, June 13, 2021
Chère Madame,

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 13, 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 ConT_EXt 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’endroit du texte de remplacement 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 13, 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 Création de nouveaux environnements

Pour créer un nouvel environnement, ConT_EXt fournit la commande `\defines-tartstop` dont la syntaxe est la suivante :

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



Dans la définition *officielle* de `\definestartstop` (voir section ??) il y a un argument supplémentaire que je n'ai pas mis ci-dessus parce qu'il est optionnel, et je n'ai pas été capable de trouver à quoi il sert. Ni le manuel d'introduction « [ConT_EXt Mark IV, an Excursion](#) », ni le manuel de référence ne l'expliquent. J'avais supposé que cet argument (qui doit être saisi entre le nom et la configuration) pouvait être le nom d'un environnement existant qui servirait de modèle initial pour le nouvel environnement, mais mes tests montrent que cette hypothèse était fausse. J'ai consulté la liste de diffusion ConT_EXt et je n'ai vu aucune utilisation de cet argument possible.

où

- **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 à exécuter avant d'entrer dans l'environnement.
 - `after` : Commandes à exécuter après avoir quitté l'environnement.
 - `style` : Style que doit avoir le texte du nouvel environnement.
 - `setups` : Ensemble de commandes créées avec `\startsetups ... \stopsetups`. Cette commande et son utilisation ne sont pas expliquées dans cette introduction.
 - `color` : Couleur à appliquer au texte
 - `inbetween`, `left`, `right` : Options non documentées que je n'ai pas réussi à faire fonctionner. D'après les tests que j'ai effectués, 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 créer avec `\Nom`. Une piste sur [la liste de diffusions NTG-context](#).



Un exemple de la définition d'un environnement pourrait être le suivant :

```

\definestartstop
[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 disparaisse 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.

3.8.1 Groupes

Un groupe est un fragment bien défini du fichier source que ConTeXt utilise 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 :

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](#)

- 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 ConT_EXt 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 ConT_EXt 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
}
```

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 ConT_EXt 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 ConT_EXt 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 T_EX 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 ConT_EXt 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 ConT _E Xt	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 ConT_EXt

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 pour-quoi, en général, elle est recommandée.

À de très rares exceptions près, nous pouvons utiliser l'unité de mesure de notre choix, car ConT_EXt 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 ConT_EXt tant que nous en connaissons le nom. Par exemple :

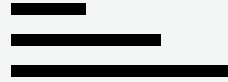
```

\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 ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt

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

L'énorme quantité de commandes et d'options de ConT_EXt 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 ConT_EXt 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 ConT_EXt, 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 ??.
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 ConT_EXt 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 ??), 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 ConT_EXt. Comme il est courant dans le monde des logiciels libres, « ConT_EXt 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 ConT_EXt 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 ConT_EXt, 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 T_EX; 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 texte simple* ou *texte 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 \LaTeX les systèmes basés sur \TeX – qui est également né aux États-Unis et a donc l'anglais comme langue maternelle – supposaient que le 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.

Con \TeX t 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'encodage 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 s'enregistrer le fichier dans le codage souhaité. Les fichiers sources destinés à être traités par Con \TeX t 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 :

- a. 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`.

- b. Indiquer à ConT_EXt dans le fichier source que l'encodage n'est pas UTF-8. Pour ce faire, nous devons utiliser la commande `\enableregime`, dont la syntaxe est `\enableregime[codage]`, où *codage* fait référence au nom par lequel ConT_EXt connaît l'encodage réel du fichier en question. Dans la [table 4.1](#), vous trouverez les différents encodages et les noms par lesquels ConT_EXt les connaît.

Codage	Nom pour ConT _E Xt	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, isoirl111, mik, mls, mnk, mos, ncc	Various

Tableau 4.1 Références des principales tables de codage pour ConT_EXt

ConT_EXt 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 `\enableregime` ConT_EXt 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 `\enableregime`, 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt. 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.  
  
Tweedledum    and    Tweedledee.
```

```
Tweedledum and Tweedledee.  
Tweedledum and Tweedledee.
```

ConT_EXt 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 ConT_EXt qui le font. Normalement, cela fonctionnera avec « `\` », c'est-à-dire un caractère `\` suivi d'un espace blanc. Mais il existe d'autres procédures qui seront examinées dans le chapitre ?? concernant les espacements horizontaux.

```
Dupont      et      Dupond.  
  
Dupont\     et\     Dupond.  
  
Dupont\ \   et\ \   Dupond.  
  
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 devons 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
rirififiLOULOU
```

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, T_EX 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. ConT_EXt 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 ConT_EXt 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	-	-	<code>\hyphen</code>
Tiret moyen / demi-cadratin	—	--	<code>\endash</code>
Tiret long / cadratin	—	---	<code>\emdash</code>
Signe moins	-	\$-\$	<code>\minus</code>

Tableau 4.2 Rules/dashes in ConT_EXt

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 T_EX, 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 ConT_EXt 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 ??).
- 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 T_EX

Afin de pouvoir travailler avec plus d'un fichier source, T_EX 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 ??.

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` ».

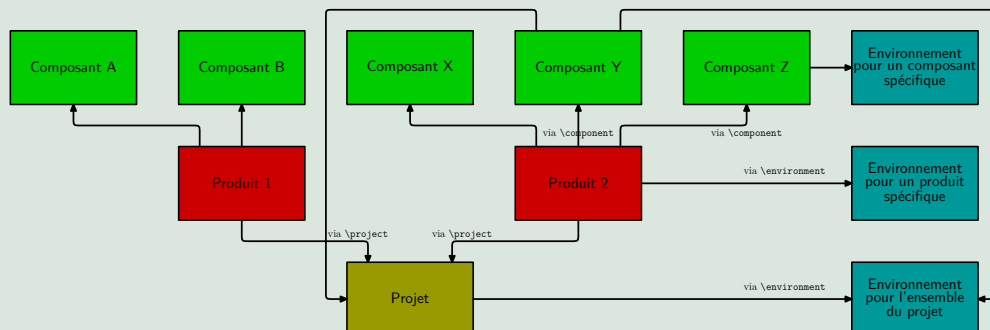
Une version plus raffinée de `\ReadFile` est `\readfile` dont le format est le suivant :

```
\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 ConT_EXt

Le troisième mécanisme que propose ConT_EXt 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'écarter de la terminologie officielle de ConT_EXt 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 ConT_EXt, 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 ConT_EXt. 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 aimerions 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     cmp_chapitre-1
  \component     cmp_chapitre-2
  \component     cmp_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 89](#).

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` *FichierEnvironnement*. 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 traductions 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 `\project 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 `\startproject` (comme pour les composants et produits)

Tout comme pour les commandes `\environnement` et `\composant`, la commande `\project` nous permet d'indiquer le nom du projet entre crochets ou de ne pas utiliser de crochets du tout. Cela signifie que `\project NomdeFichier` et `\Project[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 `\starttext` soit la commande `\environnement FichierEnvironnement` soit la commande `\project FichierProjet`.
- b. Pour un fichier produit, en insérant entre `\startproduct` et le premier `\component` soit la commande `\environnement FichierEnvironnement` soit la commande `\project 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, ConT_EXt 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 ConT_EXt 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 ??), avec une syntaxe similaire indiquée à l’option `directory` :

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

II

Global aspects of the document

Chapitre 5

Pages et pagination

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.

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 `\setuppapersize`; 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 `\setuppapersize` qui est la commande typique que l'on trouve dans le préambule du fichier source. La syntaxe *normal* de cette commande est :

```
\setuppapersize[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 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, tabloïd, 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 ConT_EXt, 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 ConT_EXt 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 ??) 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 ConT_EXt, 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 ConT_EXt 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 ConT_EXt, 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 différentes 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.

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

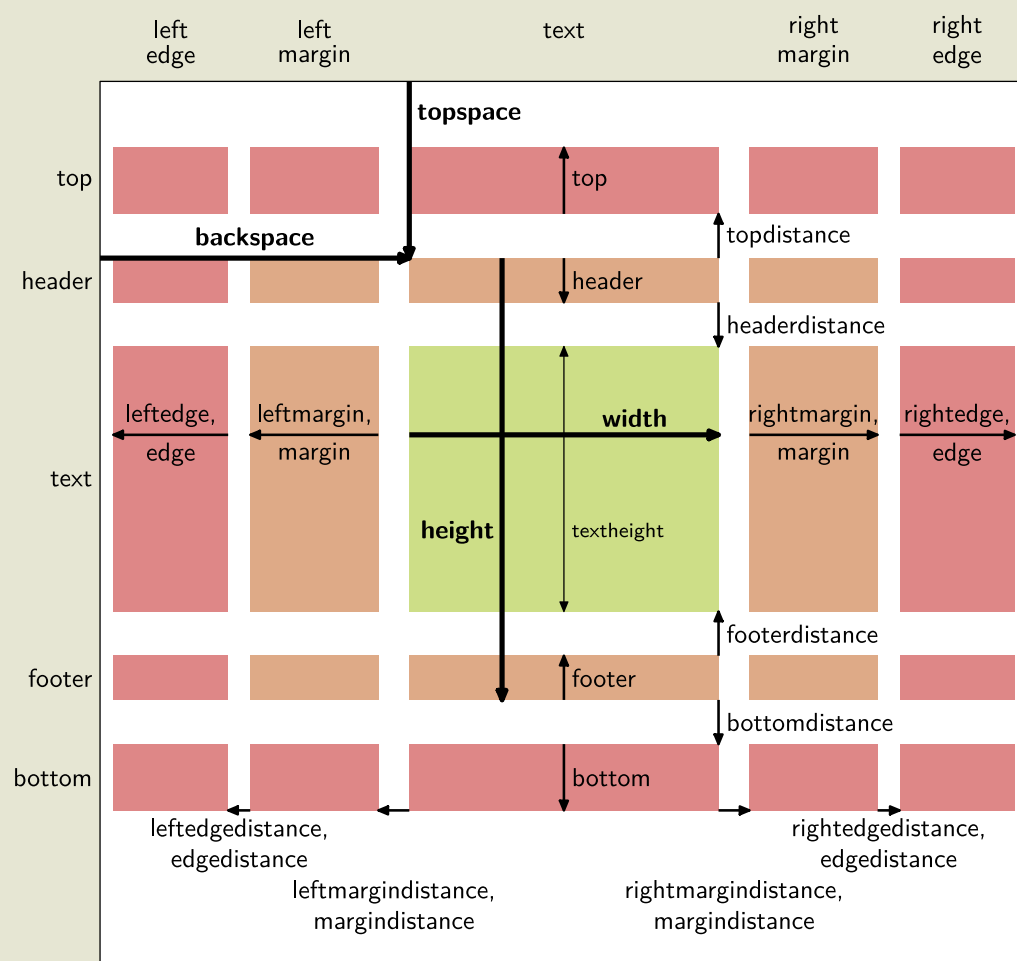


Figure 5.1 Zones et mesures sur une page

(« 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 (« outery ») ; 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**, **rightedgedistance** : 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** » + « **leftedgedistance** » + « **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 ConT_EXt 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 pourrions 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 devons 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 ??).

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 (« `single-sided` »), ou s'ils différencient 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 « `marginedge` ». 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 », « roman-numerals »), majuscules romaines (« I », « R », « Romannumerals ») 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`, `\lastpagenumber` 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 ??).

- **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 [pagenumber]` 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 `\setuppagenumbering` 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

Tweedledum

and

Tweedledee



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 ConT_EXt *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 structurale.
- **partnumber**, **chapternumber**, **sectionnumber...** : indique le numéro de la partie, du chapitre, de la section... ou de toute autre division structurale.

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 `\setupheadertexts` 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	
	<h1>1 Test</h1> <p>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, synopsisize, winnow the wheat from the chaff and separate the sheep from the goats.</p>	



	<h1>1 Test</h1> <p>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, synopsisize, winnow the wheat from the chaff and separate the sheep from the goats.</p>	
	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 :

- `\setuptoptexts` : 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`, `\atrighmargin`, `\ininner`, `\ininneredge`, `\ininnermargin`, `\inleft`, `\inleftedge`, `\inleftmargin`, `\inmargin`, `\inothet`, `\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 `\setuptoptexts` 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 `\definemargindata` avec des options particulières pour chacune.

```
\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 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` ??.
- **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 `\margin`.

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 ConT_EXt

Table of Contents: 6.1 Polices de caractères incluses dans « ConT_EXt 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 typcript pour l'utiliser partout; 6.7.5 Quelques dernières fonctionnalités avec les polices;

6.1 Polices de caractères incluses dans « ConT_EXt Standalone »

Le système de polices de ConT_EXt 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 ConT_EXt Standalone, celles présentées dans [table 6.1](#).

La colonne centrale de [table 6.1](#) indique le ou les noms par lesquels ConT_EXt 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 ConT_EXt utilise par défaut, les 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

Nom officiel	Référence ConT _E Xt	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	Emily Brontë's book
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	antypkwapoltawskiego	Emily Brontë's book
Antykwa Toruńska	antypkwatorunska	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 ConT_EXt distribution

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 ConT_EXt. Cependant, si ConT_EXt 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 ConT_EXt. 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 [ConT_EXt](#).

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 ConT_EXt. Ce dernier fait une distinction entre les polices, les styles et les variantes (ou alternatives) pour chaque style. Les *polices* incluses dans la distribution ConT_EXt (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 T_EX, 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 « sérif » 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 *romain*, 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 sérif
- Style sans sérif
- 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.

Con \TeX t 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 Con \TeX t inclut ces styles dans leur conception, et d'autre part, comme je le vois, l'écriture calligraphique est également manuscrite. Ces commandes que Con \TeX t 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 Con \TeX t 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`
- Médiéval (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 activée, les autres sont désactivées. C'est pourquoi Con \TeX t fournit des commandes pour les activer mais pas pour les désactiver ; parce que lorsque nous activons une alternative, nous désactivons celle que nous utilisons 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, ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 T_EX *Computer Modern*, conçue par Knuth, n'existait que dans certaines tailles, et a donc été réimplémentée dans une conception appelée *Latin Modern* utilisée par ConT_EXt, 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 T_EX, 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 T_EX.

6.3 Définition de la police principale du document

Par défaut, sauf si une autre police est indiquée, ConT_EXt utilisera *Latin Modern Roman* à 12 points comme police principale. Cette police a été conçue à l'origine par K_NU_TH pour être implémentée dans T_EX. 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 fioritures 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 scriptscript 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	rm, roman, serif, regular
Sans Serif	ss, sans, support, sansserif
Monospace	tt, mono, type, teletype
Manuscrite	hw, handwritten
Calligraphique	cg, calligraphic

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	\tfd
\rm	Ag	AG	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	AG	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

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

The image displays a font specimen grid for the 'tegyreadventorregular' font. The grid is organized into two main sections, each containing 16 columns and 16 rows of characters. The first section (top) covers the range from 32 to 47, and the second section (bottom) covers the range from 160 to 175. Each character is shown with its corresponding TeX character code (e.g., 32, 33, 34, etc.) and a bounding box (e.g., 848, 29, 841, 49). The characters include standard Latin letters, punctuation, and special characters. The font style is a serif typeface with a slightly irregular, hand-drawn appearance. The grid is presented on a light blue background with a white border around each character cell.

name: tegyreadventorregular at 9.0pt plane: 0 *0

6.4 Modification de la police ou de certaines de ses caractéristiques

6.4.1 Les commandes `\setupbodyfont` et `\switchto-bodyfont`

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 ConT_EXt : `\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.

`\switchobodyfont` 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`, ConT_EXt 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 ConT_EXt 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</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>\teletype</code> ,
Handwritten	<code>\hw</code> , <code>\handwritten</code> ,
Calligraphique	<code>\cg</code> , <code>\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 alternatif	Commandes qui l'active
Normal	<code>\tf</code> , <code>\normal</code>
Italique	<code>\it</code> , <code>\italic</code>
Gras	<code>\bf</code> , <code>\bold</code>
Gras-italique	<code>\bi</code> , <code>\bolditalic</code> , <code>\italicbold</code>
Oblique	<code>\sl</code> , <code>\slanted</code>
Gras-oblique	<code>\bs</code> , <code>\boldslanted</code> , <code>\slantedbold</code>
PETITES MAJUSCULES	<code>\sc</code> , <code>\smallcaps</code>
Médiéval	<code>\os</code> , <code>\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}, {\tfd 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, 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 ConT_EXt 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 : <i>Ag</i>				
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	<i>Ag</i>	AG	<i>Ag</i>	<i>Ag</i>	Ag	Ag	Ag	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	Ag	Ag
\ss	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	Ag	Ag	Ag	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	Ag	Ag
\tt	<i>Ag</i>	AG	<i>Ag</i>	<i>Ag</i>	Ag	Ag	Ag	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	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 redéfinissions la mise à l'échelle pour une taille particulière de la police principale définie par `\setupbodyfont` ou par `\switchto-bodyfont`. 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 `\switchobodyfont`. 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 co

[modern] [modern,12pt]							interlinespace
text	script	scriptscript	x	xx	small	big	
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 `\viii`¹⁴) 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*

*sé*rif 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

`\s1`. Lorsque nous utilisons ces deux dernières commandes, nous indiquons à ConT_EXt quelle alternative de police nous voulons utiliser ; mais lorsque nous utilisons `\em`, nous lui indiquons l'effet que nous voulons produire, en laissant à ConT_EXt 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 mise 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 ??.

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 :

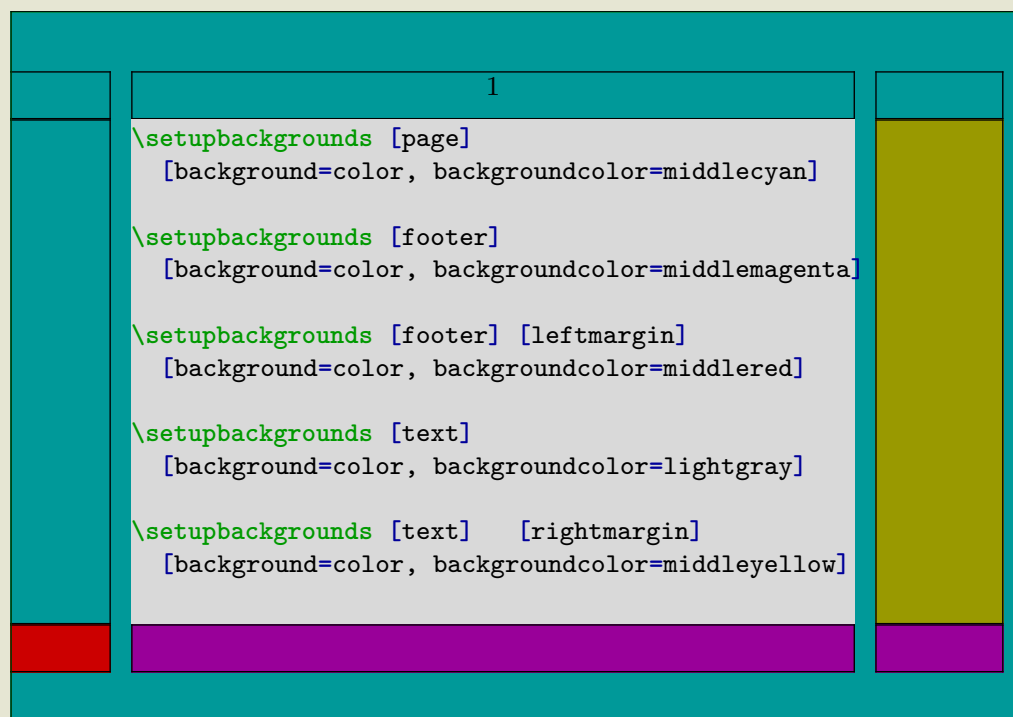
```
\definehighlight[important][color=red]
\important{Very important text}
```

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 98](#). 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

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 ?? – pour les titres de section.

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.
```

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	midldered	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 prédéfinies 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
	darkgoldenrod		r=0.720,g=0.530,b=0.040

6.6.6 Définir 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 luminosité («l» ou parfois «v», de *value*). La teinte correspond à un nombre compris entre 0 et 360 ; la saturation et la luminosité 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'exploitaion

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 ConT_EXt prenne en compte.

Dans tous les cas, ConT_EXt 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 T_EX](#). 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 ConT_EXt doit chercher les polices, en définissant la variable d'environnement « OSFONTDIR ».

- WINDOWS :

```
set OSFONTDIR=c:/windows/fonts/
```

- MAC :

```
export OSFONTDIR=/Library/Fonts:/System/Library/Fonts:$HOME/Library/Fonts
```

- GNU/LINUX :

```
export OSFONTDIR=$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
```

- 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
identifieur      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
```

- 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-graissestyle*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 « ConT_EXt Standalone ».

- 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 », « extra-light », « light », « medium », « semi-bold », « extrabold », « black », « ultrablack », « 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]
\setups[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 « sérif » du *typescript* « mynotoserif ».

```

\starttypescript [mynotoserif]
\definetypface [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]
  \setups[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]
  \definetypface [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

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	<code>\part</code>	–
2	<code>\chapter</code>	<code>\title</code>
3	<code>\section</code>	<code>\subject</code>
4	<code>\subsection</code>	<code>\subsubject</code>
5	<code>\subsubsection</code>	<code>\subsubsubject</code>
6	<code>\subsubsubsection</code>	<code>\subsubsubsubject</code>
...

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 `\subsubsubsection`, et ainsi de suite jusqu'à `\subsubsubsubsubsubsubsubsubsubsubsubsection`, ou `\subsubsubsubsubsubsubsubsubsubsubsubject`.

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 a 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 ConT_EXt 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 ConT_EXt : 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 ConT_EXt, 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. ConT_EXt 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`.

<pre>\startsection[title={Mon joli titre}] Mon texte dans \namedstructurevariable{section}{title} \stopsection</pre>	<pre>1 Mon joli titre Mon texte dans Mon joli titre</pre>
--	---

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, ConT_EXt 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 `\start-SectionName`.

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 « insidection », 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 « place-head » dont les valeurs peuvent être « yes », « no », « hidden », « empty » ou « section ». La signification des deux premiers est claire. Mais je ne suis pas si 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 comptabilisée 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 (`\SectionType [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 :

```

\defineresetset[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]
    \stopsubsection
    \startsubsection[title=Section 1.1.2]
    \stopsubsection
  \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 :

1. **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 ConT_EXt 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. ConT_EXt 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: ★ ★★ ★★★ ‡ ‡‡ ‡‡‡ ¶ ¶¶ ¶¶¶ § §§ §§§

```

2. **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]
texte
\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 ConT_EXt sur le mécanisme de numérotation des sections. Imaginons que nous souhaitons, pour faciliter 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{\tfd\bf{#1}}
\def\NiveauDeux#1{\tfc\bf{#1}}
\def\NiveauTrois#1{\tfb\bf{#1}}
\def\NiveauQuatre#1{\tfa\bf{#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 ConT_EXt 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.

- **margin** : 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 re 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-ordonnement* 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	<code>\startfrontmatter[Options] ... \stopfrontmatter</code>
Corps principal	bodypart	<code>\startbodymatter [Options] ... \stopbodymatter</code>
Annexes	appendix	<code>\startappendices [Options] ... \stopappendices</code>
Pages finales	backpart	<code>\startbackmatter [Options] ... \stopbackmatter</code>

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 les *appendices*, les sections sont numérotées en majuscules.

Il est également possible de créer de nouvelles macro-section avec `\definesectionblock`.

Chapitre 8

Table of contents, indexes, lists

Table of Contents: **8.1 Table of contents;** 8.1.1 Overall view of the table of contents; 8.1.2 Completely automatic table of contents with a title; 8.1.3 Automatic table of contents without a title; 8.1.4 Elements to incorporate in the TOC: the `criterium` option; 8.1.5 Layout of the table of contents: the `alternative` option; 8.1.6 Format of TOC entries; 8.1.7 Manual adjustments to the table of contents; **A** Including unnumbered sections in the TOC; **B** Manually adding entries to the TOC; **C** Exclude a particular section from the TOC belonging to a section type that is included in the TOC; **D** Section title text which differs in the TOC from the title in the body of the document; **8.2 Lists, combined lists and table of contents based on a list;** 8.2.1 Lists in ConTeXt; 8.2.2 Lists or indexes of images, tables and other items; 8.2.3 Combined lists; **8.3 Index;** 8.3.1 Generating the 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 Formatting the subject index; 8.3.3 Creating other indexes;

A table of contents and an index are a global aspect of a document. Almost all documents will have a table of contents, while, only some documents will have an index. For many languages (but not for English) both the table of contents and the index come under the general term «index». For English readers, a table of contents will normally come at the beginning (of a document, or possibly in some cases at the beginning of chapters as well), and the index will come at the end.

Either of these imply a particular application of the mechanism for internal references whose explanation is included in section ??.

8.1 Table of contents

8.1.1 Overall view of the table of contents

In the previous chapter we examined the commands that allow the structure of a document to be established as it has been written. This section focuses on the table of contents and the index, which in some way *mirror* the document's structure. The table of contents is very useful for getting an idea of the document as a whole (it helps contextualise information) and for searching the exact point where a particular passage might be located. Books with a very complex structure, with many sections and subsections with various levels of depth, seem to require a different kind of table of contents, since a poorly detailed one (perhaps with only the first two or three levels of sectioning) helps a lot to get an overall idea of the contents of the document, but is not very useful for locating a particular passage; unlike a very detailed table of contents, on the other hand, where it is easy to miss the forest for

the trees and lose the overall view of the document. This is why, sometimes, books with a particularly complex structure include more than one table of contents: one not too detailed at the beginning showing the main parts, and a more detailed table of contents at the beginning of each chapter as well as, perhaps, an index at the end.

These can all be generated by ConT_EXt automatically with relative ease. We can:

- Generate a complete or partial table of contents at any point in the document.
- Decide on the contents of either.
- configure their appearance down to the last detail.
- Include hyperlinks in the table of contents that allow us to jump directly to the section in question.

In fact this last utility is included by default in all tables of contents provided that the interactivity function has been enabled in the document. See, in this respect, section ??.

The explanation of this in the ConT_EXt reference manual is, in my opinion, somewhat confusing, which I think is due to the fact that too much information is introduced at once. The mechanism for building ConT_EXt tables of contents has many pieces to it; and it is difficult for a text that tries to explain them all at once to be clear. Especially for the reader who is new to the scene. By contrast, the explanation in wiki, is practically limited to examples: very useful for learning *tricks* but inadequate – I think – for understanding the mechanism and how it works. this is why the strategy I have decided to use to explain things in this introduction begins by assuming something that is not strictly true (or not completely true): that there is something in ConT_EXt called the *table of contents*. Starting with this, the *normal* commands for generating the table of contents are explained, and when these commands and their configuration are well known, I think this is the moment for introducing – though at a theoretical rather than more practical level – the information on those pieces of the mechanism that have been omitted up till then. Knowledge of these additional *pieces* allows us to create much more customised tables of contents than the ones we can call the *normal ones* created with the commands explained up to that point; however, in most cases we will not need to do this.

8.1.2 Completely automatic table of contents with a title

The basic commands for generating an automatically generated table of contents (TOC) from the numbered sections of a document (*part*, *chapter*, *section*, etc.) are `\completecontent` and `\placecontent`. The main difference between the two commands is that the first adds a *title* to the TOC; to do so, immediately before the TOC it inserts an *unnumbered chapter* whose default title is Table of Contents.

Therefore `\completecontent`:

- Inserts, at the point where it is found, a new unnumbered chapter entitled « Table of Contents ».

We recall that in ConT_EXt the command used to generate an unnumbered section at the same level as chapters, is `\title` (see [section 7.2](#)). Therefore in reality `\completecontent` does not insert a *Chapter* (`\chapter`) but a *Title* (`\title`). I have not said so because I think it may be confusing for the reader to use the names of the unnumbered section commands here, since the term *Title* also has a broader sense, and it is easy for the reader not to identify it with the concrete level of sectioning we are referring to.

- This *chapter* (actually, `\title`) is formatted exactly the same as the rest of the unnumbered chapters in the document; which by default includes a page break.
- The table of contents is printed immediately after the title.

Initially the generated TOC is *complete*, as we can deduce from the command name that generates it (`\completecontent`). But on the one hand we can limit the level of depth of the TOC as explained in [section 8.1.3](#) and, on the other, since this command is *sensitive* to the place it is found in the source file (see what is said further on about `\placecontent`), if `\completecontent` is not found at the beginning of the document it is possible that the TOC generated is not complete; and in some points of the source file it is even possible that the command is apparently ignored. If this happens, the solution is to invoke the command with the « `criterium=all` » option. Regarding this option, also see [section 8.1.3](#).

To change the default title assigned to the TOCs we use the `\setupheadtext` command whose syntax is:

```
\setupheadtext [Language] [Element=Name]
```

where *Language* is optional and refers to the language identifier used by ConT_EXt (see [section ??](#)), and *Element* refers to the element whose name we want to change (« `content` » in the case of the table of contents) and *Name* is the name or title we want to give our TOC. For example

```
\setupheadtext [en] [content=Contents]
```

will ensure that the TOC generated by `\completecontent` is entitled « Contents » instead of « Table of Contents ».

Moreover, `\completecontent` allows the same configuration options as `\placecontent`, for the explanation of which I refer to (the next section).

8.1.3 Automatic table of contents without a title

The general command for inserting a TOC without a title, generated automatically from the document's sectioning commands, is `\placecontent`, whose syntax is:

```
\placecontent [Options]
```

In principal, the table of contents will contain absolutely all numbered sections, although we can limit its level of depth with the `\setupcombinedlist` command (that we will speak of further on). So, for example:

```
\setupcombinedlist [content] [list={chapter,section}]
```

will limit the contents of the TOC to chapters and sections.

A peculiarity of this command is that it is sensitive to its location in the source file. This is very easy to explain with a few examples, but much more difficult if we want to specify exactly how the command works and which headings are included in the TOC in each case. So let's start with the examples:

- `\placecontent` placed at the beginning of the document, before the first section command (part, chapter or section, according to the situation) will generate a complete table of contents.

I am not really sure that the table of contents generated by default is *complete*, I believe it does include enough levels of sectioning to be complete in most cases; but I suspect it will not go beyond the eighth level of sectioning. In any case, as mentioned above, we can adjust the sectioning level the TOC reaches with

```
\setupcombinedlist[content][list={chapter, section, subsection, ...}]
```

- By contrast, this same command located inside a part, chapter or section will exclusively generate a TOC of the content of that element, or in other words chapters, sections and other lower levels of sectioning of a specific part, or sections (and other levels) of a specific chapter, or subsections of a specific section.

As for the technical and detailed explanation, in order to understand the default operation of `\placecontent` properly, it is essential to remember that the various sections are, in fact, *environments* for ConTeXt Mark IV that start with `\startSectionType` and end with `\stopSectionType` and can be contained within other lower level section commands. So, taking that into account, we can say that `\placecontent` generates by default a table of contents that will only include:

- Elements that belong to the *environment* (section level) where the command is placed. This means that the command when placed in a chapter will not include sections or subsections from other chapters.
- Elements that have a sectioning level lower than the level corresponding to the point where the command is located. Meaning that if the command is in a chapter, only sections, subsections and other lower levels are included; but if the command is in a section, it will be split to make the TOC of the subsection level.

Furthermore, for the table of contents to be generated, it is required that `\placecontent` be found *before* the first section of the chapter in which it is located, or before the first subsection of the section in which it is located, etc.

I'm not sure I was clear in the explanation above. Perhaps with a somewhat more detailed example than the previous ones we can better understand what I mean: let's imagine the following structure of a 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

So: `\placecontent` placed before Chapter 1 will generate a complete table of contents, similar to the one generated by `\completecontent` but without a title. But if the command is placed within Chapter 1 and before section 1.1, the table of contents will be only of the chapter; and if it is placed at the beginning of section 1.2, the table of contents will be only the content of that section. But if the command is placed, for example, between sections 1.1 and 1.2 it will be ignored. It will also be ignored if it is placed at the end of a section, or at the end of the document.

All of this, of course, refers only to the case where the command does not include options. In particular, the `criterium` option will alter that default behaviour.

Of the options allowed by `\placecontent` I will only explain two of them, the most important ones for setting up the TOC, and, moreover, the only ones that are (partially) documented in the ConTeXt reference manual. The `criterium` option, which affects the content of the TOC in relation to the place in the source file where the command is located; and the `alternative` option, which affects the general layout of the TOC to be generated.

8.1.4 Elements to incorporate in the TOC: the `criterium` option

The default operation of `\placecontent` in relation to the position of the command in the source file has been explained above. The `criterium` option alters this operation. Among others, it can take the following values:

- `all`: the TOC will be complete, regardless of the place in the source file where the command is found.
- `previous`: the TOC will only include the section commands (of the level we are at) *previous* to `\placecontent`. This option is intended for TOCS that are written at the end of the document or section in question.
- `part`, `chapter`, `section`, `subsection`...: implies that the TOC should be limited to the sectioning level indicated.
- `component`: in multifile projects (see [section 4.6](#)), it will generate only the TOC corresponding to the *component* where the `\placecontent` or `\completecontent` command is found.

8.1.5 Layout of the table of contents: the alternative option

The `alternative` option controls the overall layout of the table of contents. Its main values can be seen in [table 8.1](#).

alternative	Contents of TOC entries	Notes
a	Number – Title – Page	One line per entry
b	Number – Title – Spaces – Page	One line per entry
c	Number – Title – Leader dots – Page	One line per entry
d	Number – Title – Page	Continuous TOC
e	Title	Framed
f	Title	Left aligned, right aligned or centred
g	Title	Centred

Tableau 8.1 Ways of formatting the table of contents

The first four alternative values provide all the information of each section (its number, its title and the page number where it begins), and are therefore suitable for both paper and electronic documents. The last three alternatives only inform us about the title, so they are only suitable for electronic documents where it is not necessary to know the page number where a section begins, provided that the TOC includes a hyperlink to it, which happens by default in ConT_EXt.

Furthermore, I believe that in order to truly appreciate the differences between the various alternatives, it is best for the reader to generate a test document where he or she can analyse them in detail.

8.1.6 Format of TOC entries

We have seen that the `alternative` option of `\placecontent` or `\completecontent` allows us to control the general *layout* of the table of contents, i.e. what information will be shown for each heading, and whether or not there will be line breaks separating the different headings. Final adjustments to each TOC entry are made with the `\setuplist` command whose syntax is as follows:

`\setuplist [Element] [Configuration]`

where *Element* refers to a particular kind of section. This could be *part*, *chapter*, *section*, etc. We can also configure more than one element at the same time, separating them with commas. *Configuration* has up to 54 possibilities, many of them, as usual, not expressly documented; but this does not prevent those that are documented, or the ones that are not clear enough from allowing full adjustment of the TOC.

I will now explain the most important options, grouping them according to their usefulness, but before going into them let us remember that a TOC entry, depending on the value of the `alternative`, can have up to three different components: The section number, the title of the section, and the page number. The configuration options allow us to configure the various components globally or separately:

- *Inclusion (or not) of the different components:* If we have chosen an alternative that includes, in addition to the title, the section number and the page number (alternatives «a» «b» «c» or «d»), the options `headnumber=no` or `pagenumber=no`, it means that for the specific level we are configuring, the section number (`headnumber`) or the page number (`pagenumber`) is not displayed.
- *Colour and style:* We already know that the entry that generates a specific section in the TOC may have (depending on the alternative) up to three different components: section number, title and page number. We can jointly indicate the style and the colour for the three components using the `style` and `color` options, or do it individually for each component by means of `numberstyle`, `textstyle` or `pagestyle` (for the style) and `numbercolor`, `textcolor` or `pagecolor` for the colour.

To control the appearance of each entry, in addition to the style itself, we can apply some command to the whole entry or to one of its different elements. For this there are the `command`, `numbercommand`, `pagecommand` and `textcommand` options. The command indicated here can be a standard ConTeXt command or a command of our own creation. Section number, title text and page number will be passed as arguments to the `command` option, while the section title will be passed as an argument to `textcommand` and page number to `pagecommand`. So, for example, the following sentence will ensure that section titles are written in (fake) small caps:

```
\setuplist[section][textcommand=\Cap]
```

- *Separation of the other TOC elements:* The `before` and `after` options allow us to indicate the commands that will be executed before (`before`) and after (`after`) typesetting the TOC entry. Normally these commands are used to set either the spacing or some separating element between the previous and subsequent entries.
- *Indenting an element:* set with the `margin` option which allows us to set the amount of left indentation that the entries of the level we are configuring will have.
- *Hyperlinks embedded in the TOC:* By default the index entries include a hyperlink to the document page where the section in question begins. Using the `interaction` option we can disable this function (`interaction=no`) or limit the part of the index entry where the hyperlink will be, which can be the section number (`interaction=number` or `interaction=sectionnumber`), the section title (`interaction=text` or `interaction=title`) or the page number (`interaction=page` or `interaction=pagenumber`).
- *Other aspects:*

- `width`: specifies the separation distance between the number and title of the section. It can be a dimension, or the keyword `fit` that sets the exact width of the section number.
- `symbol`: allows the section number to be replaced by a *symbol*. Three possible values are supported: `one`, `two` and `three`. The value `none` for this option removes the section number from the TOC.
- `numberalign`: indicates the alignment of numbering elements; it can be `left`, `right`, `middle`, `flushright`, `flushleft`.

Among the multiple configuration options of the TOC, there are none that allows us to directly control the interline spacing. This will be, by default, the one that applies to the document as a whole. Often, however, it is preferable that lines in the TOC are slightly *tighter* than the rest of the document. To achieve this we should enclose the command that generates the table of contents (`\placecontent` or `\completecontent`) within of a group where a different interline spacing is established. For example:

```
\start
  \setupinterlinespace[small]
  \placecontent
\stop
```

8.1.7 Manual adjustments to the table of contents

We have already explained the two fundamental commands for generating tables of contents (`\placecontent` and `\completecontent`), as well as their options. With these two commands, TOCs are automatically generated, constructed from the existing numbered sections in the document, or in the block or segment of the document to which the table of contents refers. I will now explain certain *settings* that we can make so that the content of the TOC is not so *automatic*. This implies:

- The possibility of also including some unnumbered section titles in the TOC.
- The possibility of manually sending a particular entry to the TOC that does not correspond to the presence of a numbered section.
- The possibility of excluding a particular numbered section from the TOC.
- The possibility that the title for a particular section reflected in the TOC does not coincide exactly with the title included in the body of the document.

A. Including unnumbered sections in the TOC

The mechanism by which ConT_EXt builds the TOC means that all numbered sections are automatically included, which, as I have already said (see [section 7.4.2](#)) depends on the two (`number` and `incrementnumber`) options that we can change with `\setuphead` for each kind of section. It was also explained there that a section type where `incrementnumber=yes` and `number=no` would be an internally but not externally numbered section.

Therefore, if we want a particular unnumbered section type – for example, `title` – to be included in the TOC, we must change the value of the `incrementnumber` option for that section type, setting it to `yes` and then include that section type among those to be displayed in the TOC, which is done, as explained above, with `\setupcombinedlist`:

```
\setuphead
  [title]
  [incrementnumber=yes]

\setupcombinedlist
  [content]
  [list={chapter, title, section, subsection, subsubsection}]
```

We can then, if we wish, format this entry using `\setuplist` in exactly the same way as any of the others; for example:

```
\setuplist [title] [style=bold]
```

Note: The procedure just explained will include all instances in our document of the unnumbered section type concerned (in our example the `title` type sections). If we only wish to include a particular occurrence of that section type in the TOC, it is preferable to do so by the procedure explained below.

B. Manually adding entries to the TOC

We can send either an entry (simulating the existence of a section that does not really exist) or a command to the table of contents, from any point in the source file.

To send an entry that simulates the existence of a section that does not really exist, use the `\writetolist` whose syntax is:

```
\writetolist [SectionType] [Options] {Number}{Text}
```

in which

- The first argument indicates the level that this section entry must have in the TOC: `chapter`, `section`, `subsection`, etc.
- The second argument, which is optional, allows this entry to be configured in a particular way. If the manually sent input is omitted, it will be formatted as all the entries of the level indicated with the first argument; although, I must point out that in my tests I have not managed to make it work.

Both in the official list of ConT_EXt commands (see [section 3.6](#)) and in the wiki we are told that this argument allows the same values as `\setuplist` which is the command that allows us to format the different TOC entries. But, I insist, in my tests I have not managed to change the appearance of the TOC entry sent manually in any way.

- The third argument is supposed to reflect the numbering that the element sent to the TOC has, but I couldn't get this to work in my tests either.
- The last argument includes the text to be sent to the TOC.



This is useful, for example, if we want to send a particular unnumbered section, but only that to the TOC. In [section A](#) it explains how to get an entire category of unnumbered sections to be sent to the table of contents; but if we only want to send a particular occurrence of a section type to it, it is more convenient to use the `\writetolist` command. And so, for example, if we want the section of our document containing the bibliography not to be a numbered section, but still to be included in the TOC, we would write:

```
\subject{Bibliography}
\writetolist[section]{}{Bibliography}
```

See how we are using the unnumbered version of `section`, which is `subject`, for the section but we are sending it to the index, manually, as if it were a numbered section (`section`).

Another command intended to influence the table of contents manually is `\writebetweenlist` which is used to send not an entry itself, but a *command* to the table of contents, from a particular point in the document. For example, if we want to include a line between two items in the TOC, we could write the following at any point in the document located between the two sections concerned:

```
\writebetweenlist[section]{\hrule}
```

C. Exclude a particular section from the TOC belonging to a section type that is included in the TOC

The table of contents is constructed from *section types* established, as we already know, by the `list` option of `\setupcombinedlist`, so if a certain *section type* must appear in the TOC, there is no way of excluding a particular section from it that for whatever reasons we don't want in the TOC.

Normally, if we don't want a section to appear there, what we would do is to use its *unnumbered equivalent* meaning, for example, `title` instead of `chapter`, `subject` instead of `section`, etc. These sections are not sent to the TOC, and neither are they numbered.

However, if for any reason we want a certain section to be numbered but not appear in the table of contents, even if other types of this kind do, we can use a *trick* which consists of creating a new section type that is a clone of the section in question. For example:

```
\definehead[MySubsection][subsection]
\section{First section}
\subsection{First subsection}
\MySubsection{Second subsection}
\subsection{Third subsection}
```

This will ensure that when inserting a section type `MySubsection` the subsection counter will increase, since this section is a *clone* of the subsections, but the TOC will not be altered, since by default it does not include `MySubsection` types.

D. Section title text which differs in the TOC from the title in the body of the document

If we do not want the title of a particular section included in the TOC to be identical to the one displayed in the body of the document, we have two procedures available to us:

- Creating the section not with traditional syntax (`\SectionType{Title}`) but with `\SectionType [Options]`, or with `\startSectionType [Options]`, and assign the text we want to be written in the TOC to the `list` option (see [section 7.3](#)).
- When writing the title of the section in question in the body of the document, use the `\nolist` command: this command causes the text it takes as an argument to be replaced in the TOC by an ellipsis. For example:

```
\chapter
  [title={An \nolist{approximate and slightly repetitive}
            introduction to the reality of the obvious}]
```

would typeset as the chapter title in the body of the document, « An approximate and slightly repetitive introduction to the reality of the obvious », but would send the following text to the TOC « An ... introduction to the reality of the obvious ».

Attention: What I have just pointed out about the `\nolist` command is stated in both the ConT_EXt reference manual and the [wiki](#). For me, however, it produces a compiling error, telling me that the `\nolist` command is undefined.

8.2 Lists, combined lists and table of contents based on a list

Internally, for ConT_EXt, a table of contents is nothing more than a *combined list*, which, in turn, as its name suggests, consists of a combination of simple lists. Therefore the basic notion from which ConT_EXt builds the table of contents is that of a list. Several lists are combined to form a table of contents. By default, ConT_EXt contains a predefined combined list called « content » and this is what the commands examined so far work with: `\placecontent` and `\completecontent`.

8.2.1 Lists in ConT_EXt

In ConT_EXt, a *list* is a range of numbered elements about which we need to remember three things:

1. The number.
2. The name or title.
3. The page where it is found.

This happens with numbered sections; but also with other elements of the document such as images, tables, etc. In general, those elements for which there is a command whose name begins with `\place` which places them as `\placetable`, `\placefigure`, etc.

In all these cases, ConTeXt automatically generates a list of the different times the type of element in question appears, its number, title and page. Thus, for example, there is a list of chapters, called `chapter`, another of sections, called `section`; but also another of tables (called `table`) or images (called `figure`). Lists generated automatically by ConTeXt are always called the same as the item they store.

A list will also be automatically generated if we create, for example, a new type of numbered section: when we create it we will be implicitly creating also the list that stores them. And if for a non-numbered section by default, we set the option `incrementnumber=yes`, making it a numbered section, we will also be implicitly creating a list with that name.

Together with the implicit lists (automatically defined by ConTeXt) we can create our own lists with `\definelist`, whose syntax is

```
\definelist[ListName][Configuration]
```

Items on the list are added:

- In lists predefined by ConTeXt, or created by it as a result of creating a new floating object (see section ??), automatically each time an item from the list is inserted into the document, either by a sectioning command or by the `\placeWhatever` command for other types of lists, for example: `\placefigure`, will insert any image in the document, but it will also insert the corresponding entry in the list.
- Manually in any kind of list with `\writetolist[ListName]`, already explained in subsection B of section 8.1.7. The `\writebetweenlist` command is also available. It too was explained in that section.

Once a list has been created and all its items included in it, the three basic commands related to it are `\setuplist`, `\placelist` and `\completelist`. The first allows us to configure what the list looks like; the last two insert the list in question at the point in the document where it finds them. The difference between `\placelist` and `\completelist` is similar to the difference between `\placecontent` and `\completecontent` (see sections 8.1.2 and 8.1.3).

So, for example,

```
\placelist[section]
```

will insert a list of the sections, including a hyperlink to them if the document's interactivity is enabled and if, in `\setuplist`, we have not set `interaction=no`. A list of sections is not exactly the same as a table of contents based on sections: the idea of a table of contents usually includes the lower levels as well (sub-sections, subsubsections, etc.). But a list of sections will include only the sections themselves.

The syntax of these commands is:

```
\placelist [ListName] [Options]
```

```
\setuplist [ListName] [Configuration]
```

The `\setuplist` options have already been explained in [section 8.1.6](#), and the options for `\placelist` are the same as for `\placecontent` (see [section 8.1.3](#)).

8.2.2 Lists or indexes of images, tables and other items

From what has been said so far, it can be seen that, since ConTeXt automatically creates a list of images placed in a document with the `\placefigure` command, generating a list or index of images at a particular point in our document is as simple as using the `\placelist[figure]` command. And if we want to generate a list with a title (similar to what we get with `\completecontent`) we can do it with `\completelist[figure]`. We can do similarly with the other four predefined kinds of floating objects in ConTeXt: tables (« table »), Graphics (« graphic »), *intermezzos* (« intermezzo ») and chemical formulas (« chemical »), although for specific cases of these, ConTeXt already includes a command that generates them without a title: (`\placelistoffigures`, `\placelistoftables`, `\placelistofgraphics`, `\placelistofintermezzi` and `\placelistofchemicals`), and another that generates them with a title: (`\completelistoffigures`, `\completelistoftables`, `\completelistofgraphics`, `\completelistofintermezzi` and `\completelistofchemicals`), in a similar way to `\completecontent`.

In the same way, for floating objects we ourselves have created (see [section ??](#)) the `\placelistof<FloatName>` and `\completelistof<FloatName>` will be automatically created.

For lists we have created with `\definelist` we can create an index with `\placelist [ListName]` or with `\completelist [ListName]`.

8.2.3 Combined lists

A combined list is, as its name suggests, a list that combines items from different previously defined lists. By default, ConTeXt defines a combined list for tables of content whose name is « content », but we can create other combined lists with `\definecombinedlist` whose syntax is:

```
\definecombinedlist [Name] [Lists] [Options]
```

where

- *Name*: is the name the new combined list will have.
- *Lists*: refers to the names of lists to be combined, separated by commas.
- *Options*: Configuration options for the list. They can be indicated at the time of defining the list, or, probably preferably, when the list is invoked. The main

options (which have already been explained) are `criterium` (subsection 8.1.4 of section 8.1.3) and `alternative` (in subsection 8.1.5 in the same section).

A collateral effect of creating a combined list with `\definecombinedlist` is that it also creates a command called `\placeListName` which serves to invoke the list, that is: to include it in the output file. So for example,

```
definecombinedlist [TOC]
```

will create the command `\placeTOC`; and

```
definecombinedlist [content]
```

will create the command `\placecontent`

But wait, `\placecontent`! Isn't this the command that is used to create a *normal* table of contents? Indeed: this means that the standard table of contents is actually created by ConT_EXt by means of the following command:

```
\definecombinedlist
[content]
[part, chapter, section, subsection,
 subsubsection, subsubsubsection,
 subsubsubsubsection]
```

Once our combined list is defined, we can configure it (or reconfigure it) with `\setupcombinedlist` which allows the already explained options `criterium` (see subsection 8.1.4 in section 8.1.3) and `alternative` (see subsection 8.1.5 in the same section), as well as the `list` option to *change* the lists included in the combined list.

The official list of ConT_EXt commands (see section 3.6) does not mention the `list` option among the options allowed for `\setupcombinedlist`, but it is used in several examples of the use of this command in the wiki (which, moreover, does not mention it in the page devoted to this command either). I have also checked that the option works.

8.3 Index

8.3.1 Generating the index

A subject index consists of a list of significant terms, usually located at the end of a document, indicating the pages where such a subject can be found.

When books were put typeset by hand, generating a subject index was a complex task, as well as a tedious one. Any change in the pagination could affect all the entries in the index. Therefore, they were not very common. Today, the computer mechanisms for typesetting mean that, while the task is likely to continue being tedious, it is no longer so complex given that it is not so difficult for a computer system to maintain an up-to-date list of data associated with an index entry.

To generate a subject index we need:

1. Determine which words, terms or concepts are to be part of it. This is a task that only the author can do.
2. Check at which points in the document each entry in the future index appears. Although, to be precise, more than *checking* the places in the source file where the concept or issue is discussed, what we do when we work with ConT_EXt is to *mark* those spots, inserting a command that will then serve to generate the index automatically. This is the tedious part.
3. Finally, we generate and format the index by placing it at the point of our choice in the document. The latter is quite simple with ConT_EXt and requires only one command: `\placeindex`.

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 ConT_EXt.

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{\backslashslash 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]{\backslashslash 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 `\completeindex` 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 `\completeindex` 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 `\completeindex` previously inserts a new chapter in the final document, called « Index » by default, inside which the index will be typeset.

8.3.2 Formatting the subject index

Subject indexes are a particular application of a more general structure ConTeXt calls « *register* »; therefore the index is formatted with the command:

```
\setupregister[index] [Configuration]
```

With this command we can:

- Determine what the index will look like with its different elements. Namely:
 - The index headings which are usually letters of the alphabet. By default these are in lower case. With `alternative=A` we can set them to be in upper case.
 - The entries themselves, and their page number. The appearance depends on the `textstyle`, `textcolor`, `textcommand` and `deeptextcommand` options for the actual entry, and `pagestyle`, `pagecolor` and `pagecommand`, for the page number. With `pagenumber=no` we can also generate a subject index without page numbers (although I don't know if this could be useful).
 - The `distance` option measures the width of separation between the name of an entry and the page numbers; but it also measures the amount of indentation for subentries.

The names of the `style`, `textstyle`, `pagestyle`, `color`, `textcolor`, and `pagecolor` options are clear enough to tell us what each one does I think. For `command`, `pagecommand`, `textcommand` and `deeptextcommand`, I refer to the explanation for similarly named options in [section 7.4.4](#), regarding the configuration of section commands.

- To set the general appearance of the index, which includes, among others the commands to execute before (`before`) or after (`after`) the index, the number of columns it needs to have (`n`), whether the columns should be equal or not (`balance`), the alignment of entries (`align`), etc.

8.3.3 Creating other indexes

I have explained the subject index as if only one such index would be possible in a document; but the truth is that documents can have as many indexes as desired. There could be an index of personal names, for example, which collects the names of people mentioned in the document, with an indication of the place where they are cited. These are still a kind of index. In a legal text we could also create a special index for mentions of the Civil Code; or, in a document like the present one, an index of macros explained in it, etc.

To create an additional index in our document we use the `\defineregister` command whose syntax is:

```
\defineregister[IndexName] [Configuration]
```

where *IndexName* is the name the new index will have, and *Configuration* controls how it works. It is also possible to configure the index later on by means of

```
\setupregister[IndexName] [Configuration]
```

Once a newly named index *IndexName* has been created we will have the `\IndexName` command at our disposal to mark the entries that this index will have in a similar way to the way entries are marked with `\index`. The `seeIndexName` command also lets us create entries that refer to other entries.

For example: we could create an index of ConT_EXt commands in this document with the command:

```
\defineregister[macro]
```

that would create the `\macro` command. This lets me mark all the references to ConT_EXt commands as an index entry, and then generate the index with `\placemacro` or `\completemacro`.

Creating a new index enables the `\IndexName` command to mark its entries, and the `\placeIndexName` and `\completeIndexName` commands for generating the index. But these latter two commands are actually abbreviations of two more general commands applied to the index in question. Thus, `\placeIndexName` is equivalent to `\placeregister[IndexName]` and `\completeIndexName` is equivalent to `\completeregister[IndexName]`.

III

Particular issues

Appendices

Annexe A

Index

a

aide et ressources 28

c

compilation 35

composition 16

d

define 61

definestartstop 63

e

encode 73

espace 76

espace vide 76

f

fichier

composant 87

environnement 85

produit 87

projet 88

fichiers

chemin 90

fonts 140

OSFONTDIR 140

l

langages de balisage

balises 17

markup 17

m

Mark II 15

Mark IV 15

macro-structure 106

moteurs T_EX 19

mtxrun 140

p

page

dimension 93

préambule 38

r

rédaction 16

règles syntaxiques

commande 59

repertoire

chemin 90

s

saut de ligne 78

setupwhitespace 69

structure

chemin 90

composant 87

produit 87

projet 88

t

tabulation 76

tirets 79

trait d'union 79

