

Une courte (?) introduction à ConT_EXt Mark IV

Une courte (?) introduction à ConT_EXt Mark IV

Version 1.6 [14 juin 2021]

© 2020-2021, Joaquín Ataz-López

Titre original: Una introducción (no demasiado breve) a ConT_EXt 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](#).
- 14 juin 2021: Version 1.6 : Corrections suggérées après une nouvelle lecture du document, à l’occasion de sa traduction en anglais

Table des matières

Préface	8
I Qu'est ce que ConT_EXt et comment travailler avec ?	16
1 ConT_EXt : une vue d'ensemble	18
1.1 Qu'est-ce que ConT _E Xt ?	18
1.2 La composition typographique de document	20
1.3 Les langages de balisage	22
1.4 T _E X et ses dérivés	24
1.5 ConT _E Xt	27
2 Notre premier fichier source	36
2.1 Préparation de l'expérience outils nécessaires	36
2.2 L'expérience elle-même	39
2.3 La structure de notre fichier d'exemple	44
2.4 Quelques détails supplémentaires sur la façon d'exécuter « context » ..	45
2.5 Traitement des erreurs	47
3 Les commandes et autres concepts fondamentaux de ConT_EXt ..	52
3.1 Les caractères réservés de ConT _E Xt	53
3.2 Les commandes à proprement parler	57
3.3 Périmètre des commandes	60
3.4 Options de fonctionnement des commandes	64
3.5 Résumé sur la syntaxe des commandes et des options, et sur l'utilisation des crochets et des accolades lors de leur appel.	68
3.6 La liste officielle des commandes ConT _E Xt	70
3.7 Définir de nouvelles commandes	71
3.8 Autres concepts fondamentaux	76
3.9 Méthode d'auto apprentissage pour ConT _E Xt	81
4 Fichiers sources et projets	84
4.1 Codage des fichiers sources	84
4.2 Caractères dans le(s) fichier(s) source(s) que ConT _E Xt traite d'une manière spéciale	87
4.3 Projet simple et projet multi-fichiers	92
4.4 Structure du fichier source d'un projet simple	93
4.5 Gestion multi-fichiers à la T _E X	95
4.6 Gestion multi-fichiers à la ConT _E Xt	98

II	Composition des éléments généraux au document	104
5	Pages et pagination	106
5.1	Taille de la page	106
5.2	Éléments sur la page	111
5.3	Mise en page (<code>\setuplayout</code>)	114
5.4	Numérotation des pages	119
5.5	Sauts de page forcés ou suggérés	122
5.6	En-têtes et pieds de page	124
5.7	Insertion d'éléments de texte dans les bords de page et les marges	129
6	Polices d'écriture et couleurs dans ConTeXt	132
6.1	Polices de caractères incluses dans « ConTeXt Standalone »	132
6.2	Caractéristiques d'une police	134
6.3	Définition de la police principale du document	137
6.4	Modification de la police ou de certaines de ses caractéristiques	141
6.5	Autres questions relatives à l'utilisation de styles alternatifs	148
6.6	Utilisation et configuration des couleurs	150
6.7	Bonus 1 - Utilisation des polices du système d'exploitation	156
7	Structure du document	162
7.1	Les divisions structurelles d'un document	162
7.2	Types et hiérarchie des sections	164
7.3	Syntaxe commune des commandes liées aux sections	166
7.4	Format et configuration des sections et de leurs titres	168
7.5	Définir de nouvelles commandes de section	184
7.6	La macrostructure du document	185
8	Table des matières, index, listes	188
8.1	Table des matières	188
8.2	Listes, listes combinées et tables des matières basées sur une liste	204
8.3	Index	208
9	Références et hyperliens	214
9.1	Types de référence	214
9.2	Références internes	216
9.3	Documents électroniques interactifs	224
9.4	Hyperliens vers des documents externes	226
9.5	Création de signets dans le PDF final	230
9.6	Pièces jointes	231
III	Composition des éléments locaux	232
10	Caractères, mots, texte et espace horizontal	234
10.1	Obtenir des caractères qui ne sont pas normalement accessibles à partir du clavier	234

10.2	Special character formats	243
10.3	Character and word spacing	247
10.4	Compound words	250
10.5	The language of the text	251

I Appendices 258

A Index des commandes 260

B Index 268

I

TXT

ConT_EXt

1 - Panorama

2 - Premier doc

3 - Commandes

4 - Fichier source

PDF

II

5 - Pagination

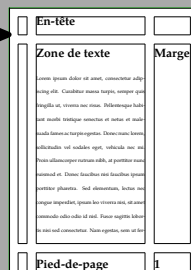
5.1 - Taille de la page

5.3 - Mise en page

5.4 - Numérotation pages

5.6 - En-têtes et pieds de page

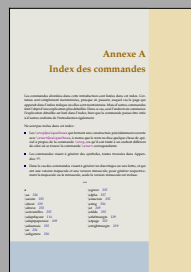
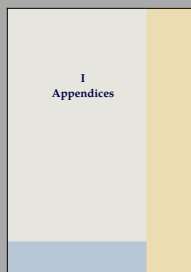
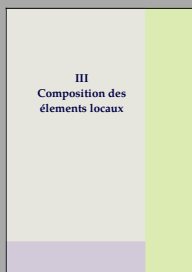
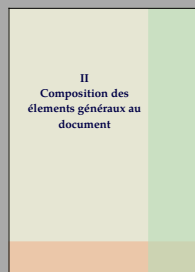
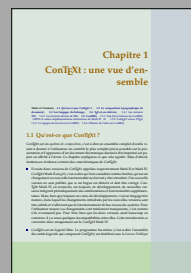
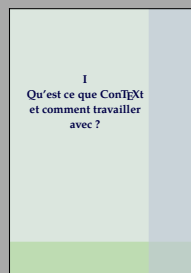
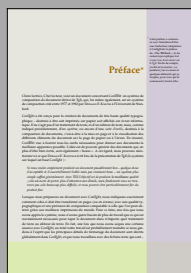
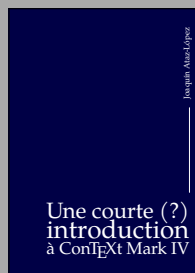
5.7 - Marges



6 - Polices
de caractère
et couleurs

7 - Structure du document

7.6 - Macro-structure



8 - Table
des matières,
index, listes

9 - Références
et hyperliens

III

10 - Lettres,
mots et
espaces
horizontaux

?? -
Paragraphes,
lignes et
espaces
verticaux

?? -
Constructions
spéciales et
paragraphes

?? - Images,
tableaux et
autres
flottants

Préface*

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

Chère lectrice, Cher lecteur, voici un document concernant ConT_EXt 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.

ConT_EXt 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, ConT_EXt 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 ConT_EXt) :

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 ConT_EXt, 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 ConT_EXt, 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 ConT_EXt, et que nous travaillons avec des fichiers texte qui sont –

² Au moment de la première version de ce texte, ceci était vrai ; mais au printemps 2020, le Wiki ConTeXt a été mis à jour et nous devons supposer qu'à partir de ce moment la distribution « officielle » de ConTeXt est devenue LMTX. Cependant, pour ceux qui entrent dans le monde de ConTeXt pour la première fois, je recommande quand même d'utiliser « ConTeXt Standalone » puisque c'est une distribution plus stable. L'annexe ?? explique comment installer l'une ou l'autre des distributions.

³ Pour la liste, voir section ??.

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

Documentation

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

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

En 2013, ce manuel [a été partiellement mis à jour](#) mais beaucoup de ses sections n'ont pas été réécrites et il contient des informations relatives à la fois à ConTeXt Mark II et ConTeXt Mark IV (la version actuelle), sans toujours préciser clairement quelles informations se rapportent à chacune des versions. C'est peut-être la raison pour laquelle ce manuel ne se trouve pas parmi les documents inclus dans « ConTeXt Standalone ». Pourtant, malgré ces défauts, et une fois lu « [ConTeXt Mark IV, an Excursion](#) », le manuel reste le meilleur document pour continuer à apprendre ConTeXt. Autres informations également toujours très utiles pour démarrer avec ConTeXt, celles contenues le sur [ConTeXt Garden wiki](#), qui, au moment où nous écrivons ces lignes, est plein remaniement et présente une structure beaucoup plus claire, bien qu'elles 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 « [ConTeXt Commands](#) »³ qui comprend, pour chacune d'elles, l'ensemble des options de

configuration possibles mais ne précise pas quelles commandes ne fonctionnent que dans l'une ou l'autre des versions.

Fondamentalement, cette introduction a été rédigée en s'inspirant des quatre sources d'information énumérées préalablement : « [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 ressurent 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

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.



les sujets sur lesquels je n'ai pas trouvé d'informations ou que je n'ai pas pu (ou voulu) vérifier personnellement ; parce que parfois le résultat de mon test n'était pas concluant, d'autres fois parce que je n'ai pas toujours tout essayé : le nombre de commandes et d'options de 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 ConT_EXt met à la disposition de l’auteur pour la rédaction de documents bien structurés. Le [Chapitre 7](#) se concentre sur la structure elle-même (divisions structurelles du document) et le [Chapitre 8](#) sur la valorisation de cette structure dans une table des matières.
- Enfin, le [Chapitre 9](#) se concentre sur l’aspect clé de l’utilisation de références par un document, références vers d’autres points du même document (références internes) mais aussi vers des documents externes (références externes). Dans cette dernière situation, nous n’aborderons que le cas de références impliquant un *liens* vers le document externe. Ce chapitre explique l’utilisation de l’outil de ConT_EXt 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 10](#) et suivants) se concentre sur des aspects plus concrets, plus spécialisés. Non seulement les chapitres sont maintenant indépendants les uns des autres, mais également les sections les unes des autres au sein d’un même chapitre (sauf, peut-être, dans le dernier chapitre). Étant donné la grande quantité de fonctionnalités proposées par ConT_EXt 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 ConT_EXt je n’ai considéré que les chapitres suivants :
- Les chapitres [10](#) 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](#). I 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
Eté 2020

I

**Qu'est ce que ConTExT
et 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 30 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 fonctionnalité 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.

⁵ ConT_EXt 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 «ConT_EXt» avec son logo (ConT_EXt) 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.

- 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 :
 - 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 ConT_EXt à quoi doit ressembler le document final généré à partir des fichiers texte originaux. L'ensemble complet des instructions ConT_EXt constitue en fait un *langage* ; et puisque ce langage permet de *programmer* la transformation typographique d'un texte, on peut dire que ConT_EXt 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 ConT_EXt, nous devons donc faire la différence entre le document que nous rédigeons et le document que ConT_EXt 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 ConT_EXt 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,

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.

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

- Dans des textes formatés typographiquement, comme le présent texte, en utilisant le logo que j'ai utilisé jusqu'à présent : Les trois lettres sont en majuscules, avec le «E» central légèrement décalé vers le bas pour faciliter un rapprochement entre le «T» et le «X» ; c'est-à-dire : «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».

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

Cette convention est suivie dans tous les dérivés de $\text{T}_{\text{E}}\text{X}$ qui l’incluent dans leur propre nom, comme par exemple $\text{ConT}_{\text{E}}\text{Xt}$, qui lorsqu’il est écrit en mode texte doit être écrit « ConTeXt ».

1.4.1 Les moteurs $\text{T}_{\text{E}}\text{X}$

Le programme $\text{T}_{\text{E}}\text{X}$ 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 « $\text{T}_{\text{E}}\text{X}$ ». 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 $\text{T}_{\text{E}}\text{X}$* (engine en anglais). En dehors du programme original, les principaux moteurs $\text{T}_{\text{E}}\text{X}$ sont, par ordre chronologique d’apparition $\text{pdfT}_{\text{E}}\text{X}$, $\varepsilon\text{-T}_{\text{E}}\text{X}$, $\text{X}_{\text{E}}\text{T}_{\text{E}}\text{X}$ et $\text{LuaT}_{\text{E}}\text{X}$. Chacun d’entre eux est censé intégrer les améliorations de son prédécesseurs. Ces améliorations, en revanche, jusqu’à l’apparition de $\text{LuaT}_{\text{E}}\text{X}$, 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 $\text{T}_{\text{E}}\text{X}$ à utiliser fait l’objet d’un débat animé dans l’univers $\text{T}_{\text{E}}\text{X}$. Je ne m’y attarderai pas ici, car $\text{ConT}_{\text{E}}\text{Xt}$ Mark IV ne fonctionne qu’avec $\text{LuaT}_{\text{E}}\text{X}$. En fait, dans le monde de $\text{ConT}_{\text{E}}\text{Xt}$ la discussion sur les moteurs devient une discussion sur l’utilisation de Mark II (qui fonctionne avec $\text{pdfT}_{\text{E}}\text{X}$ et $\text{X}_{\text{E}}\text{T}_{\text{E}}\text{X}$) ou Mark IV (qui fonctionne avec $\text{LuaT}_{\text{E}}\text{X}$).

1.4.2 Les formats dérivés de $\text{T}_{\text{E}}\text{X}$

Le noyau, ou cœur, de $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$ qui imprime votre logo ($\text{\textcolor{violet}{T}\textcolor{violet}{e}\textcolor{violet}{X}}$), 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 « $\text{\textcolor{violet}{T}\textcolor{violet}{e}\textcolor{violet}{X}}$ » 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. Con \TeX t 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 exécute 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 ConT_EXt

En fait, si ConT_EXt a commencé comme un *format* de T_EX, aujourd'hui il est beaucoup plus que cela. ConT_EXt comprend :

1. Un très large ensemble de macros de T_EX. Si Plain T_EX 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_NUTH 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 mélangent 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 documentation « 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt.

1.5.2 ConT_EXt versus L^AT_EX

Comme le format dérivé de T_EX le plus populaire est L^AT_EX la comparaison entre celui-ci et ConT_EXt 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 ConT_EXt 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, ConT_EXt 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, ConT_EXt 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 incluons, 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 illibiles 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 tagguées « 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 ConT_EXt. 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 ConT_EXt, est facilement personnalisable et, comme c'est l'éditeur généralement utilisé par les développeurs de ConT_EXt « ConT_EXt 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 ConT_EXt et « ConT_EXt 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 ConT_EXt.

2. **Une distribution ConT_EXt** 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 ConT_EXt 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 ConT_EXt 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.mkiv
mtx-context      | current version: 2020.03.10 14:44
mtx-context      | main context file: /usr/share/texmf/tex/context/base/mkiv/context.mxl
mtx-context      | current version: 2020.03.10 14:44
```

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

3. **Un programme de visualisation de fichier PDF** afin de visualiser le résultat de notre expérience à l'écran. Sur les systèmes Windows et Mac OS, la visionneuse omniprésente est Adobe Acrobat Reader. Il n'est pas installé par défaut (ou ne l'était pas lorsque j'ai cessé d'utiliser Microsoft Windows, il y a plus de 15 ans). L'installation se fait la première fois que vous essayez d'ouvrir un fichier PDF, il est donc généralement déjà installé. Sur les systèmes Linux/Unix, il n'y a pas de version mise à jour d'Acrobat Reader, mais il n'est pas nécessaire non plus, car il existe littéralement des dizaines de très bons 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 inexistantes dans l'alphabet anglais (comme les voyelles accentuées ou les eñes) ou de ne pas permettre de sélectionner le texte, d'envoyer le document à l'imprimante ; c'est juste un 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 ConTeXt... Bienvenue dans le monde du *logiciel libre* !

2.2 L'expérience elle-même

A. Rédaction du fichier source

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

```
% Première ligne du document

\mainlanguage[fr]    % Langue français

\setuppapersize[S5]  % Format du papier

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

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

\starttext           % Début du contenu du document

\startchapter
[title=La maison sur le port]

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

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

\stopchapter

\stoptext           % Fin du document
```



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

B. Encodage du fichier

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

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

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

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

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

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

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

D. Traitement du document source

Pour l'étape suivante, après s'être assuré que ConT_EXt 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 devrions donc vérifier chacune de ces possibilités, et les corriger, jusqu'à ce que la compilation se déroule correctement.

1 La maison sur le port

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

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

Figure B.1 La maison sur le port

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

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

2.3 La structure de notre fichier d'exemple

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

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

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

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

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

```
$ context --help
```

Parmi les options les plus intéressantes, citons les suivantes :

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

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

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

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

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

version indique la version de ConTeXt.

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

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

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

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

Pour les premières utilisation et pour l'apprentissage de 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 ConTeXt. Nous devons y chercher une ligne qui commence par un avertissement d'erreur « `tex error` », pour cela nous pouvons utiliser la fonction de recherche de texte de l'éditeur. Nous trouverons les lignes d'erreur suivantes :

```

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

1.14 \starttext
                                % Début du contenu du document

```

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

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

Pensez que les ordinateurs sont très bons et très rapides pour exécuter des instructions, mais très maladroits pour lire nos pensées, et que le mot « starttext » n'est pas le même que « starttext ». 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 pourrions pas le modifier et nous devrions continuer à enquêter jusqu'à ce que nous découvrions les raisons de l'erreur et sa possible solution.

Dans la pratique, lorsqu'on crée un document relativement volumineux avec 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 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 KNUTH, le père

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.

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.

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 rappel d'une idée à compléter ou à réviser, d'une illustration à construire.

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

; ou pour empêcher la compilation d'un extrait dont nous ne sommes pas sûrs, mais sans le supprimer du fichier source au cas où nous voudrions y revenir plus tard ; ou pour partager des commentaires lros 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 cloture un groupe préalablement ouvert avec « { ».
- # Ce caractère est utilisé pour définir les macros. Il fait référence aux arguments de la macro. Voir [section 3.7.1](#) dans ce chapitre.
- ~ Introduit un espace blanc insécable dans le document pour éviter un saut de ligne entre les mots qu'il sépare, ce qui signifie que deux mots séparés par le caractère ~ resteront toujours sur la même ligne. Nous parlerons de cette instruction et de l'endroit où elle doit être utilisée dans section ??.
- | 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 considérera comme

commentaire. L'utilisation incorrecte de la « \ » peut également produire un résultat inattendu si elle ou les caractères qui la suivent immédiatement constituent une commande connue de 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.



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.

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 TEXBook », il a pris l'habitude de représenter les espaces vides importants par le symbole « `_` ». Ainsi, par exemple, si nous voulions montrer que deux mots du fichier source doivent être séparés par deux espaces vides, nous écririons « `word1_word2` ».

- 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 important : le nom de la commande ne peut être composé que de lettres ; les chiffres, les signes de ponctuation ou tout autre type de symbole ne sont pas autorisés. Seules les lettres minuscules ou majuscules sont autorisées. N'oubliez pas, par ailleurs, que ConTeXt fait une distinction entre les minuscules et les majuscules, ce qui signifie que les commandes `\mycommand` et `\MyCommand` sont différentes. Mais `\MaCommande1` et `\MaCommande2` seraient considérées comme identiques, puisque n'étant pas des lettres, «1» et «2» ne font pas partie du nom des commandes.



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

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

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

```
Connaître \TeX aide à l'apprentissage de \ConTeXt.
```

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

```
Connaître \TeX      aide à l'apprentissage de \ConTeXt.
```

```
Connaître \TeX{} aide à l'apprentissage de \ConTeXt.
```

```
Connaître \TeX\ aide à l'apprentissage de \ConTeXt.
```

```
Connaître TEXaide à l'apprentissage de ConTEXt.
```

```
Connaître TEX, si non indispensable, aide à l'apprentissage de ConTEXt.
```

```
Connaître TEXaide à l'apprentissage de ConTEXt.
```

```
Connaître TEX aide à l'apprentissage de ConTEXt.
```

```
Connaître TEX aide à l'apprentissage de ConTEXt.
```

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

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

- Les caractères réservés « `{}` », utilisé à la quatrième ligne de l'exemple. Le caractère réservé « `{` », comme je l'ai dit, ouvre un groupe, et « `}` » ferme un groupe, donc la séquence « `{}` » introduit un groupe vide. Un groupe vide n'a aucun effet sur le document final, mais il aide `ConTeXt` à savoir que le nom de la commande qui le précède est terminé. On peut aussi créer un groupe autour de la commande en question, par exemple en écrivant « `{\TeX}` ». Dans les deux cas, le résultat sera que la première « non-lettre » après `\TeX` n'est pas un espace vide.
- Le symbole de contrôle « `\` » (une barre oblique inverse suivie d'un espace vide, voir la note sur [page 57](#)) utilisé à la cinquième ligne de l'exemple. L'effet de ce symbole de contrôle est d'insérer un espace blanc dans le document final. Pour

bien comprendre la logique de ConT_EXt, il peut être utile de prendre le temps de voir ce qui se passe lorsque ConT_EXt rencontre un mot de contrôle (par exemple `\TeX`) suivi d'un symbole de contrôle (par exemple « `_` ») :

- ConT_EXt 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 T_EX 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 T_EX) 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 T_EX et est trop complexe pour être expliquée ici. Mais cela fonctionne : une fois cette commande définie, chaque fois que nous utilisons `\txt-`, ConT_EXt la remplace par `\TeX` en éliminant le trait d'union, mais en l'utilisant en interne pour savoir que le nom de la commande est déjà terminé, de sorte qu'un espace blanc immédiatement après ne serait pas supprimé.

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

3.3 Périmètre des commandes

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

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

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

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

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

In *The T_EXBook*, K_NUTH explained **everything you need to know** about T_EX.

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

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

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

Tweedledum and Tweedledee

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

¹¹ test

Notez que, bien que dans le premier groupe de commandes (celles qui ne requièrent pas d’argument), les accolades sont parfois utilisées pour déterminer le champ d’action, mais cela n’est pas nécessaire pour que la commande fonctionne. La commande est conçue pour être appliquée à partir du point où elle apparaît. Ainsi, lorsque vous déterminez son champ d’application en utilisant des crochets, la commande est placée *entre ces crochets*, contrairement au deuxième groupe de commandes, où les parenthèses encadrant le texte auquel la commande doit être s’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, ConTeXt a un *environnement* sans nom commençant par la commande `\start` (aucun autre texte ne suit ; juste *start*, c'est pourquoi je l'appelle un *environnement sans nom*) et se termine par la commande `\stop`. Je pense que la seule fonction de cette commande est de créer un groupe.

Je n'ai lu nulle part dans la documentation de ConTeXt que l'un des effets des environnements est de grouper leur contenu, mais c'est le résultat de mes tests avec un certain nombre d'environnements 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'incluons les commandes nécessaires à la création du groupe (voir [section 3.8.1](#)).

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

3.4 Options de fonctionnement des commandes

3.4.1 Commandes qui peuvent fonctionner de différentes façon distinctes

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

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

```
\framed{Tweedledum}
```

Tweedledum

et celui d'une version personnalisée :

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

Tweedledum

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

```

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

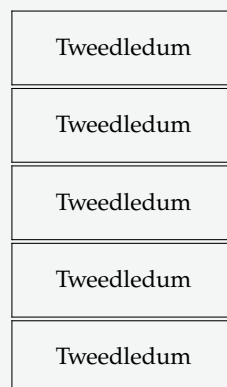
\framed[width=3cm, height=1cm]{Tweedledum}

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

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

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

```



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

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

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

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

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

reconfigurer le fonctionnement de `\framed`, en le faisant avec `\setupframed`. Ainsi,

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

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

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

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

En continuant avec l'exemple du `\framed`, il est évident que si notre document utilise plusieurs types de cadres, chacun avec des mesures différentes, l'idéal serait de pouvoir *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 ConTeXt

- Les commandes commencent toujours par le caractère « \ ».
- Certaines commandes peuvent prendre un ou plusieurs arguments.
- Les arguments qui indiquent à la commande *comment* elle doit fonctionner ou qui affectent son fonctionnement d'une manière ou d'une autre, sont introduits entre crochets.
- Les arguments qui indiquent à la commande sur quelle partie du texte elle doit agir sont présentés entre accolades.

Lorsque la commande ne doit agir que sur une seule lettre, comme c'est le cas, par exemple, de la commande `\buildtextcedilla` (pour donner un exemple – le « ç » si souvent utilisée en catalan), les accolades autour de l'argument peuvent être 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 formes 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 conseillé 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 ConTeXt.

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

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

où

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

Marseille, June 16, 2021
Chère Madame,

Anne Smith
Consultant

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

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

Anne Smith
Consultant

Marseille, June 16, 2021
Chère Madame Dupond,

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}
```

Anne Smith
Consultant

Marseille, June 16, 2021
Cher Monsieur Antoine Dupond,

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 « [ConTeXt 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 fautive. J'ai consulté la liste de diffusion ConTeXt 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.
 - `setup` : Ensemble de commandes créées avec `\startsetup ... \stopsetup`. 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\noindeatation,
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.

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

3.8.1 Groupes

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

- 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 T_EX ces groupes ne pouvaient pas être ouverts et fermés avec des accolades, car cela aurait généré des accolades déséquilibrées dans le fichier source, ce qui aurait toujours provoqué une erreur lors de la compilation.

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

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

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

Ainsi, si nous traitons l'exemple suivant

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

B C B

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

Une autre utilisation possible des groupes concerne les commandes ou instructions conçues pour s'appliquer exclusivement au caractère qui est écrit après elles. Dans

ce cas, si nous voulons que la commande s'applique à plus d'un caractère, nous devons inclure dans un groupe les caractères auxquels nous voulons que la commande ou l'instruction s'applique. Ainsi, par exemple, le caractère réservé « `^` » qui, nous le savons déjà, convertit le caractère suivant en exposant lorsqu'il est utilisé dans l'environnement mathématique ; ainsi, si nous écrivons, par exemple, « `4^2x` », nous obtiendrons « 4^2x ». Mais si nous écrivons « `4^{2x}` », nous obtiendrons « 4^{2x} ».

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

3.8.2 Dimensions

Bien que nous puissions utiliser ConTeXt parfaitement sans nous soucier des dimensions, nous ne pourrions pas utiliser toutes les possibilités de configuration sans leur accorder une certaine attention. Car, dans une large mesure, la perfection typographique atteinte par 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 ConTeXt les dimensions sont indiquées par un nombre décimal suivi par l'unité de mesure. Les unités qui peuvent être utilisées se trouvent dans table 3.2.

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

Tableau 3.2 Unités de mesure dans ConTeXt

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

puisque tout est mis en cohérence avec la taille de la police de caractère. C’est 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 in-
terne 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 `\Ma-
DimensionA10mm` sans espace entre le nom de la mesure et sa valeur.

Toutefois, l’attribution d’une valeur directement à une mesure interne est consi-
dérée comme une « inelegant ». En général, il est recommandé d’utiliser les com-
mandes 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 ty-
pographiques.

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` (indi-
quant 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 ConTeXt

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

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

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

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

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

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

Pour savoir si ces commandes existent ou non, consultez la liste officielle des commandes (voir section ??), 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 une commande similaire à la première à partir d'un clone*. 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 ConT_EXt; 4.6.1 Fichiers d'environnement; 4.6.2 Composants et produits; 4.6.3 Projets ConT_EXt; 4.6.4 Aspects communs des environnements, composants, produits et projets;

Comme nous le savons déjà, lorsque nous travaillons avec ConT_EXt nous commençons toujours par un fichier texte dans lequel sont incluses, outre le contenu du texte, un certain nombre d'instructions indiquant à ConT_EXt 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 :

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

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

Tableau 4.1 Références des principales tables de codage pour ConTeXt

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

Avec `\enableregime` ConTeXt inclut la commande `\useregime` qui prend en argument une ou plusieurs références de codage. Je n'ai trouvé aucune information sur cette commande ni sur la façon dont elle diffère de `\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 10.3](#) 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 ConTeXt nous pouvons utiliser un seul fichier source qui inclut absolument tout le contenu de notre document final ainsi que tous les détails s’y rapportant, dans ce cas nous parlons de « projet simple », ou, au contraire, nous pouvons utiliser un certain nombre de fichiers sources qui partagent le contenu de notre document final, et dans ce cas nous parlons de « projet multi-fichier ».

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

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

4.4 Structure du fichier source d'un projet simple

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

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

```
% Première ligne du document

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

\starttext    % Début du contenu à proprement parler

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

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

Figure 4.1 Fichier source contenant un projet simple

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

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

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

- les paramètres pour nos propres macros ([section 3.7](#)).
- Etc.

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

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

4.5 Gestion multi-fichiers à la T_EX

Afin de pouvoir travailler avec plus d'un fichier source, T_EX a inclus la primitive appelée `\input`, qui fonctionne également dans ConT_EXt, 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 ConT_EXt 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. ConT_EXt le cherchera dans le répertoire de travail, et s'il ne le trouve pas là, il le cherchera dans les répertoires inclus dans la variable de l'environnement TEXROOT. Si le fichier n'est finalement pas trouvé, il produira une erreur de compilation.

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

```
\input MaConfiguration    % Commandes de configuration générale

\starttext

    \input PageDeTitre
    \input Preface

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

\stoptext
```

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

```
\input MaConfiguration    % Commandes de configuration générale

\starttext

% \input PageDeTitre
% \input Preface

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

\stoptext
```

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

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

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

4.5.2 `\ReadFile` et `\readfile`

Comme nous venons de le voir, si ConT_EXt 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, ConT_EXt 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` ».

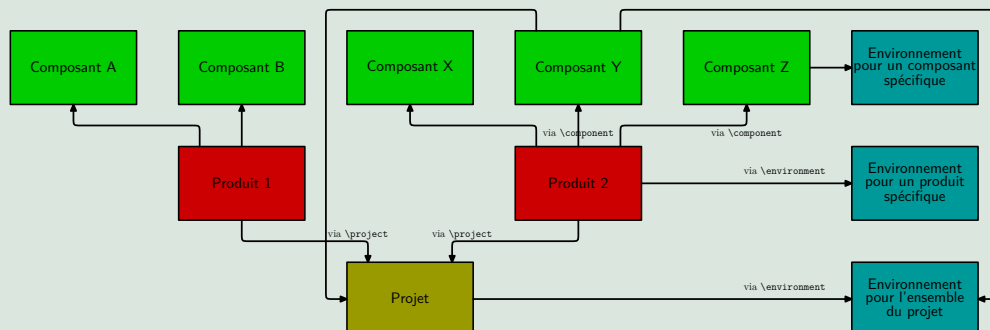
Une version plus raffinée de `\ReadFile` est `\readfile` dont le format est le suivant :

```
\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 102](#).

Notez que le contenu réel de notre document sera réparti entre les différents fichiers «composant» 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

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

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

Selon le wiki, il existe d'autres valeurs possibles pour le deuxième argument : « sous-format », « double-format » et « double-surformat ». Mais lors de mes propres tests, je n'ai constaté

aucun changement après l'utilisation de l'une de ces valeurs ; la définition officielle de la commande (voir section ??) 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 Con-TeXt 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.

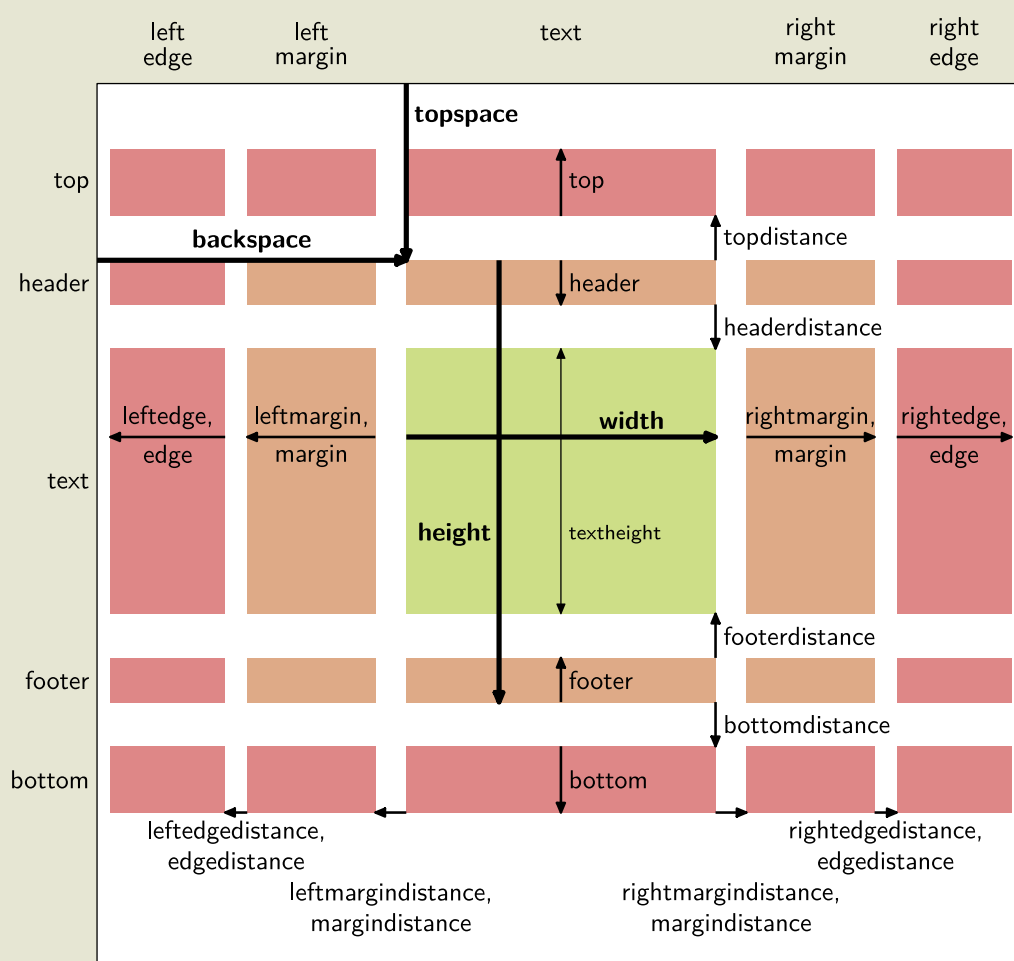


Figure 5.1 Zones et mesures sur une page

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

- **Edges (Bords)** : espace blanc entourant la zone de texte. Bien que la plupart des traitements de texte les appellent « margins », il est préférable d'utiliser la terminologie de ConT_EXt car elle nous permet de faire la différence entre les bords en tant que tels, où il n'y a pas de texte (bien que dans les documents électroniques, il puisse y avoir des boutons de navigation et autres), et les marges où peuvent parfois se trouver certains éléments de texte, comme, par exemple, les notes de marge.

- La hauteur du bord supérieur est contrôlée par deux mesures : le bord supérieur lui-même (« top ») et la distance entre le bord supérieur et l'en-tête (« topdistance »). La somme de ces deux mesures est appelée « topspace ».
- La taille des bords gauche et droit dépend de la « leftedge » « rightedge » respectivement. Si l'on souhaite que les deux soient de la même longueur, on peut les configurer simultanément avec l'option « edge ».

Dans les documents destinés à être imprimés en recto-verso, on ne parle pas de bords gauche et droit mais de bords intérieur (« inner ») et extérieur (« 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. ConT_EXt appelle uniquement les marges latérales (gauche et droite) des marges. Les marges sont situées entre le bord et la zone de texte principale et sont destinées à accueillir certains éléments de texte comme, par exemple, les notes de marge, les titres de section ou leurs numéros.

Les dimensions qui contrôlent la taille des marges sont :

- **margin** : utilisé lorsque l'on souhaite définir simultanément les marges à la même taille.
- **leftmargin**, **rightmargin** : définit la taille des marges gauche et droite respectivement.
- **edgedistance** : Distance entre le bord et la marge.
- **leftgedistance**, **rightgedistance** : Distance entre le bord et les marges gauche et droite respectivement.
- **margindistance** : Distance entre la marge et la zone de texte principale.
- **leftmargindistance**, **rightmargindistance** : Distance entre la zone de texte principal et les marges droite et gauche respectivement.

- **backspace** : cette mesure représente l'espace entre le coin gauche du papier et le début de la zone de texte principale. Elle est donc constituée de la somme de « **leftedge** » + « **leftedgedistance** » + « **leftmargin** » + « **leftmarginindistance** ».
- **En-tête (Header) et pied de page (footer)** : Ils contiennent généralement des informations qui permettent de contextualiser le texte, comme le numéro de page, le nom de l'auteur, le titre de l'ouvrage, le titre du chapitre ou de la section, etc. Dans ConTeXt ces zones de la page sont affectées par les dimensions suivantes :
 - **header** : Hauteur de l'en-tête.
 - **footer** : Hauteur du pied de page
 - **headerdistance** : Distance entre l'en-tête et la zone de texte principale de la page.
 - **footerdistance** : Distance entre le pied de page et la zone de texte principale de la page.
 - **topdistance**, **bottomdistance** : Ces deux éléments ont été mentionnés précédemment. Il s'agit de la distance entre le bord supérieur et l'en-tête ou le bord inférieur et le pied de page, respectivement.
- **Zone de texte principal** : Il s'agit de la zone la plus large de la page, contenant le texte du document. Sa taille dépend des variables « **width** » et « **textheight** ». La variable « **height** », quant à elle, mesure la somme de « **header** », « **headerdistance** », « **textheight** », « **footerdistance** » et « **footer** ».

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

5.3 Mise en page (`\setuplayout`)

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

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

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

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

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

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

- En ajustant « `topspace` ».
 - En donnant les valeurs de « `fit` » ou « `middle` » à « `height` ». Ces valeurs fonctionnent de la même manière que dans le cas de la largeur. Le premier calcule la hauteur en fonction du reste des composants, et le la seconde rend les marges supérieure et inférieure égales, puis calcule la hauteur du texte.
 - Une fois que « `height` » est ajusté, en ajustant la hauteur de l’en-tête ou du pied de page si nécessaire, sachant que dans ce cas « `hauteur du texte` » sera automatiquement réajustée.
- Une autre possibilité pour déterminer de manière indirecte la hauteur de la 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, ConT_EXt utilise les chiffres arabes pour la numérotation des pages et le numéro apparaît centré dans l'en-tête. Pour modifier ces caractéristiques, ConT_EXt 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 ConT_EXt 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 numéroté 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt.

- 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 ConT_EXt 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 T_EX appelle une *boîte verticale*.

Au début de ce document (sur page ??), j'ai indiqué qu'en interne, tout est une *boîte* pour T_EX. La notion de boîte est fondamentale dans T_EX 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 T_EX 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 ConT_EXt 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, synopsise, 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, synopsise, 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
`\inouter`, où l'on constate que
le texte est dans la marge ex-
terne et justifié vers l'interne

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

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

- **location** : indique dans quelle marge le texte sera placé. Elle peut être `left`, `right` ou, dans les documents recto-verso, `outer` ou `inner`. Par défaut, il s'agit de `left` pour les documents recto et de `outer` pour les documents recto-verso.
- **width** : largeur disponible pour l'impression du texte. Par défaut, la largeur totale de la marge sera utilisée.
- **margin** : indique si le texte sera placé dans la `margin` elle-même ou dans le `edge`, voire directement au contact du texte principal avec `normal`.
- **align** : alignement du texte. Les mêmes valeurs sont utilisées ici que dans `\setupalign` ??.
- **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 typecript 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

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	<i>Emily Brontë's book</i>
Tex Gyre Pagella	pagella, palatino	Emily Brontë's book
Tex Gyre Termes	termes, times	Emily Brontë's book
DejaVu	dejavu dejavu-condensed	Emily Brontë's book Emily Brontë's book
Gentium	gentium	Emily Brontë's book
Antykwa Poltawskiego	antykwapoltawskiego	Emily Brontë's book
Antykwa Toruńska	antykwaterunska	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

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

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

Les polices utilisées par Windows ne sont pas des *logiciels libres* (en fait, presque rien dans Windows n'est un *logiciel libre*), elles ne peuvent donc pas être incluses dans une distribution de 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 T_EX 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.

ConT_EXt 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 ConT_EXt inclut ces styles dans leur conception, et d'autre part, comme je le vois, l'écriture calligraphique est également manuscrite. Ces commandes que ConT_EXt 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 ConT_EXt 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 activé, les autres sont désactivés. C'est pourquoi ConT_EXt 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 ConT_EXt 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, ConTeXt 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`.

<code>\setupbodyfont[modern,17.3pt]</code> Coucou	Coucou
<code>\setupbodyfont[modern,17.5pt]</code> Coucou	Coucou
<code>\definebodyfontenvironment[17.5pt]</code> <code>\setupbodyfont[modern,17.5pt]</code> 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 : <i>Ag</i>													
	<code>\tf</code>	<code>\sc</code>	<code>\sl</code>	<code>\it</code>	<code>\bf</code>	<code>\bs</code>	<code>\bi</code>	<code>\tfx</code>	<code>\tfxx</code>	<code>\tfa</code>	<code>\tfb</code>	<code>\tfc</code>	<code>\tfd</code>
<code>\rm</code>	<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>	<i>Ag</i>	<i>Ag</i>
<code>\ss</code>	<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>	<i>Ag</i>	<i>Ag</i>
<code>\tt</code>	<i>Ag</i>	<i>AG</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>

[adventor] [adventor,14pt] \mr : <i>Ag</i>													
	<code>\tf</code>	<code>\sc</code>	<code>\sl</code>	<code>\it</code>	<code>\bf</code>	<code>\bs</code>	<code>\bi</code>	<code>\tfx</code>	<code>\tfxx</code>	<code>\tfa</code>	<code>\tfb</code>	<code>\tfc</code>	<code>\tfd</code>
<code>\rm</code>	<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>	<i>Ag</i>	<i>Ag</i>
<code>\ss</code>	<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>	<i>Ag</i>	<i>Ag</i>
<code>\tt</code>	<i>Ag</i>	<i>AG</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>	<i>Ag</i>

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]`

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
848	29 841	21 842	22 843	23 844	24 845	25 846	26 847	27 850	28 851	29 852	29 853	29 854	2c 855	2d 856	2e 857	2f 858
	0	1	2	3	4	5	6	7	8	9	:	:	<	=	>	?
060	30 861	31 862	32 863	33 864	34 865	35 866	36 867	37 870	38 871	39 872	3a 873	3b 874	3c 875	3d 876	3e 877	3f 878
	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
100	40 881	41 882	42 883	43 884	44 885	45 886	46 887	47 888	48 889	49 890	4a 891	4b 892	4c 893	4d 894	4e 895	4f 896
	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
120	50 897	51 898	52 899	53 900	54 901	55 902	56 903	57 904	58 905	59 906	5a 907	5b 908	5c 909	5d 910	5e 911	5f 912
	ı	q	b	c	d	e	f	g	h	i	j	k	l	m	n	o
140	60 913	61 914	62 915	63 916	64 917	65 918	66 919	67 920	68 921	69 922	6a 923	6b 924	6c 925	6d 926	6e 927	6f 928
	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
160	70 929	71 930	72 931	73 932	74 933	75 934	76 935	77 936	78 937	79 938	7a 939	7b 940	7c 941	7d 942	7e 943	
	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
240	a0 241	a1 242	a2 243	a3 244	a4 245	a5 246	a6 247	a7 250	a8 251	a9 252	aa 253	ab 254	ac 255	ad 256	ae 257	af 258
	ö	±	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿	
260	b0 261	b1 262	b2 263	b3 264	b4 265	b5 266	b6 267	b7 270	b8 271	b9 272	ba 273	bb 274	bc 275	bd 276	be 277	bf 278
	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
300	c0 301	c1 302	c2 303	c3 304	c4 305	c5 306	c6 307	c7 310	c8 311	c9 312	ca 313	cb 314	cc 315	cd 316	ce 317	cf 318
	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
320	d0 321	d1 322	d2 323	d3 324	d4 325	d5 326	d6 327	d7 330	d8 331	d9 332	da 333	db 334	dc 335	dd 336	de 337	df 338
	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
340	e0 341	e1 342	e2 343	e3 344	e4 345	e5 346	e6 347	e7 350	e8 351	e9 352	ea 353	eb 354	ec 355	ed 356	ee 357	ef 358
	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ
360	f0 361	f1 362	f2 363	f3 364	f4 365	f5 366	f6 367	f7 370	f8 371	f9 372	fa 373	fb 374	fc 375	fd 376	fe 377	ff 378

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 `\switchtobodyfont`

Pour changer la police, le style ou la taille, nous pouvons utiliser la même commande avec laquelle nous avons établi la police au début du document, lorsque nous ne voulons pas utiliser la police par défaut de 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.

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

Cocou Cocou

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

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

A. Changement de style et de style alternatif

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

Style	Commandes qui l'active
Romain	<code>\rm</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]%
{\tfx test}, {\tfx test}, {\tf test}, {\tfa test}, {\tfb test}, {\tfc
test}, {\tfd test}

{\tfx 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 `\tfx` `\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>				
	<code>\tf</code>	<code>\sc</code>	<code>\sl</code>	<code>\it</code>	<code>\bf</code>	<code>\bs</code>	<code>\bi</code>	<code>\tfx</code>	<code>\tfxx</code>	<code>\tfa</code>	<code>\tfb</code>	<code>\tfc</code>	<code>\tfd</code>
<code>\rm</code>	<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
<code>\ss</code>	<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
<code>\tt</code>	<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 `\switchobdyfont`. 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*

sérif autrement (par exemple, en français, nous pourrions l'appeler « lineale » ou « sansempattement »), nous pouvons créer des synonymes en écrivant :

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

coucou coucou

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

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

coucou **coucou**

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

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

6.5 Autres questions relatives à l'utilisation de styles alternatifs

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

6.5.1 Italique, oblique et mise en valeur

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

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

D'où l'exemple suivant :

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

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

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

Un autre avantage de `\em` est qu'il ne s'agit pas d'un style alternatif, donc il ne désactive pas l'alternative que nous avions auparavant et donc, par exemple, dans un texte qui est en gras, avec `\em` nous obtiendrons du gras oblique sans avoir besoin de faire explicitement appel à `\bs`. De même, si la commande `\bf` apparaît dans un texte qui est déjà 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 10.2.1](#).

6.6 Utilisation et configuration des couleurs

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

6.6.1 Utiliser des couleurs pour des éléments textuels

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

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

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

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

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

Very important text

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

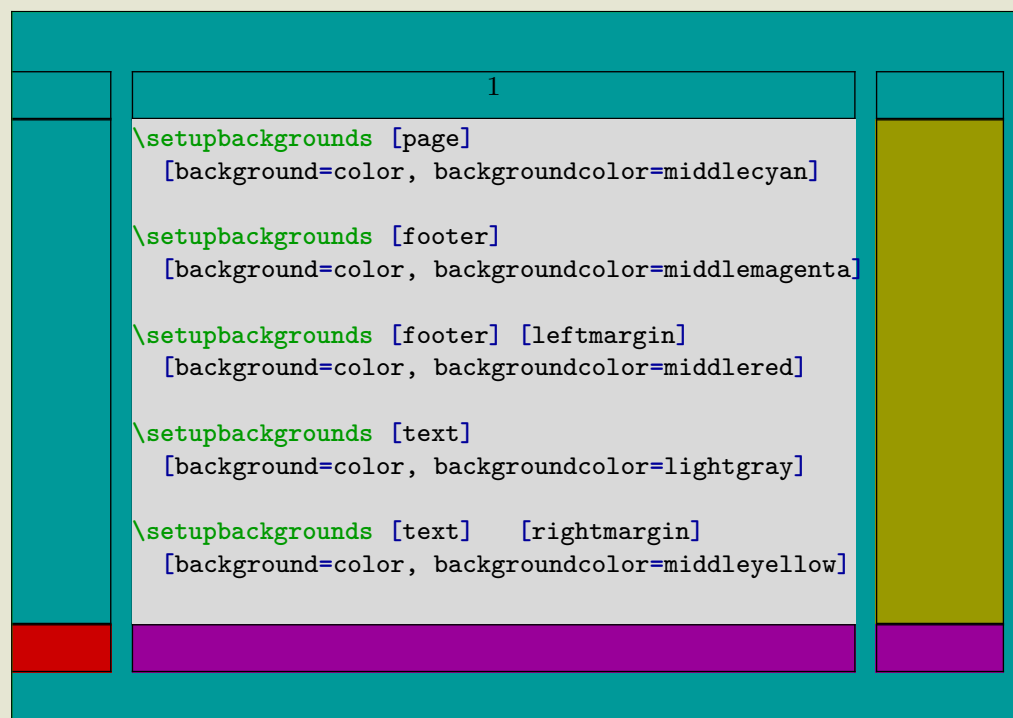
```
\definehighlight[important][color=red]  
\important{Very important text}
```

Very important text

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

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

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



Cette commande définit la couleur de fond des pages comme étant le cyan, et vous voyez comment il est possible d’affecter une couleur à chaque zone de la page vue à la figure [section 5.1 page 111](#). Comme valeur pour « backgroundcolor », nous pouvons utiliser le nom de l’une des couleurs prédéfinies ([section 6.6.4](#)).

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

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

Texte coloré

6.6.3 Utiliser des couleurs pour des portions de texte

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

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

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

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

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

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

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

6.6.4 Couleurs prédéfinies

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

Nom	Tonalité claire	Tonalité moyenne	Tonalité foncée
black			
white			
gray	lightgray	middlegray	darkgray
red	lightred	midddlered	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
white black	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 luminescence («l» ou parfois «v», de *value*). La teinte correspond à un nombre compris entre 0 et 360 ; la saturation et la luminescence doivent être un nombre décimal compris entre 0 et 1. Par exemple :

```
\definecolor [CouleurD] [h=43.00, s=0.89,
v=0.38]
\color[CouleurD]{Texte Couleur D.}
```

Texte Couleur D.

5. **Couleurs HWB** : Le modèle HWB est une norme suggérée pour CSS4 qui mesure la teinte («h», de *hue*), et le niveau de blanc («w», de *whiteness*) et de noir («b», de *blackness*). La teinte correspond à un nombre compris entre 0 et 360, tandis que la blancheur et la noirceur sont représentées par un nombre décimal compris entre 0 et 1.

```
\definecolor [CouleurE] [h=43.00, w=0.04,
b=0.28]
\color[CouleurE]{Texte Couleur E.}
```

Texte Couleur E.

6. **Couleur échelle de gris** : basé sur un composant appelé («s», de *scale*) qui mesure la quantité de gris. Il doit s'agir d'un nombre compris entre 0 et 1. Par exemple :

```
\definecolor [CouleurF] [s=0.65] %
\color[CouleurF]{Texte Couleur F.}
```

Texte Couleur F.

Il est également possible de définir une nouvelle couleur à partir d'une autre couleur. Par exemple, la couleur dans laquelle les titres sont écrits dans cette introduction est définie comme suit

```
\definecolor [CouleurG] [0.8(orange)] %
\definecolor [CouleurH] [0.6(orange)] %
\definecolor [CouleurI] [0.4(orange)] %
\definecolor [CouleurJ] [0.2(orange)] %
\color[CouleurG]{Texte Couleur G.} \\
\color[CouleurH]{Texte Couleur H.} \\
\color[CouleurI]{Texte Couleur I.} \\
\color[CouleurJ]{Texte Couleur J.}
```

Texte Couleur G.
Texte Couleur H.
Texte Couleur I.
Texte Couleur J.

6.7 Bonus 1 - Utilisation des polices du système d'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 ConT_EXt 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 installée sur votre ordinateur, et que vous avez déjà mis à jour les bases de données ConTeXt comme indiqué précédemment, allez directement au point 2.

Sinon, vous devez d'abord la télécharger et la stocker. Le site de Google fournit un fichier zip avec les 4 variations alternatives (Regular 400, Regular 400 italic, Bold 700, Bold 700 italic).

1. Stockez-les dans un dossier dédié indexé par ConTeXt (voir ci-dessus).
 - par exemple, créez un répertoire « Noto-serif » dans la distribution ConTeXt « tex/texmf-fonts/fonts/ » (ou bien, sous LINUX, dans « /.fonts »).
 - dézippez et stockez les fichiers .ttf dans « tex/texmf-fonts/fonts/Noto-serif/ ».
 - Régénérer les bases de données ConTeXt

```
mtxrun --generate
mtxrun --script font --reload
```

2. Maintenant vous pouvez vérifier le nom de la police utilisé pour identifier les polices, en lançant le script mtxrun :

```
mtxrun --script fonts --list --all --pattern=*notoserif
identifier      familyname  fontname    filename    subfont
instances
notoserif       notoserif   notoserif    NotoSerif-Regular.ttf
notoserifbold   notoserif   notoserifbold NotoSerif-Bold.ttf
notoserifbolditalic notoserif   notoserifbolditalic NotoSerif-BoldItalic.ttf
notoserifitalic notoserif   notoserifitalic NotoSerif-Italic.ttf
```

3. Vous pouvez maintenant utiliser la police n'importe où dans vos fichiers sources avec la commande `\definedfont[name:lefontname*default]` (il est bon d'ajouter « *default » pour bénéficier des fonctionnalités par défaut, comme par exemple le crénage (kerning).

```
\definedfont [name:notoserifbolditalic*default at 12 pt]%
Le renard brun et rapide saute par-dessus le chien paresseux.
```

Le renard brun et rapide saute par-dessus le chien paresseux.

6.7.3 Utilisation de divers styles alternatifs de police

Il n'est pas agréable de devoir écrire `\definedfont[name:mapolice-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 ».

1. Définissez un nouveau *typescript* dans votre fichier d'entrée, avec `\starttypescript`.
 - Définissez les liens entre les noms de fichiers et les noms lisibles par le public avec `\definefontsynonym`.
 - Dans cet exemple, le *typescript* s'appelle « mynotoserif ».
 - Rappel : vous trouvez les noms de fichiers pour les polices Noto Serif avec « `mtxrun --script fonts --list --all --pattern=*notoserif` »

```
\starttypescript [mynotoserif]
% \definefontsynonym[Human readable]      [file:filename without extension]
\definefontsynonym[NotoSerif-Regular]     [file:NotoSerif-Regular]
\definefontsynonym[NotoSerif-Italic]       [file:NotoSerif-Italic]
\definefontsynonym[NotoSerif-Bold]         [file:NotoSerif-Bold]
\definefontsynonym[NotoSerif-BoldItalic]   [file:NotoSerif-BoldItalic]
\stoptypescript
```

C'est ici que vous pouvez identifier notamment les alternatives « thin », « extra-light », « light », « medium », « semi-bold », « extrabold », « black », « ultrablack », « condensed », « extracondensed ».

2. L'étape ennuyeuse, définir les liens entre les noms de base ConT_EXt 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 ConT_EXt).

```
\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 ConT_EXt.
- 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

qqdeqqqqq

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 ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 ConT_EXt 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.

ConT_EXt 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, ConT_EXt 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 `\subsubsubsubsection`, et ainsi de suite jusqu'à `\subsubsubsubsubsubsubsubsubsubsubsubsubsection`, ou `\subsubsubsubsubsubsubsubsubsubsubsubsubject`.

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`.

```
\startsection[title={Mon joli
titre}]
Mon texte dans
\namedstructurevariable{section}{title}
\stopsection
```

```
1 Mon joli titre
Mon texte dans Mon joli titre
```

Le fait, pour l'utilisateur, de disposer ou accéder à des variables permet des utilisations très avancées de ConTeXt en raison du fait que des décisions peuvent être prises concernant la compilation ou non d'un fragment, ou de quelle manière le faire, ou avec quel modèle en fonction de la valeur d'une variable particulière. Ces utilitaires ConTeXt, cependant, dépassent le cadre du matériel que je souhaite traiter dans cette introduction.

7.4 Format et configuration des sections et de leurs titres

7.4.1 Les commandes `\setuphead` and `\setupheads`

Par défaut, 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` ConT_EXt fournit la commande `\setupheads` qui affecte globalement toutes les commandes de section. Le wiki ConT_EXt 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 ConT_EXt, bien que ce ne soit probablement pas dit au bon endroit : certaines options ne fonctionnent que si nous utilisons la syntaxe `\startSectionName`.

Cette information est contenue en relation avec `\setupheads`, mais pas avec `\setuphead` qui est pourtant l'endroit où la majeure partie des options sont expliquées et où, si cela ne doit être dit qu'à un seul endroit, cela il serait le plus raisonnable de l'indiquer. D'autre part, l'information ne mentionne que l'option « inside section », 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 ConT_EXt 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 `ConTeXt` ont un nombre très limité, il n'est approprié d'utiliser ce type de numérotation que lorsque le nombre maximal à atteindre n'est pas trop élevé. `ConTeXt` prévoit quatre jeux de symboles différents : `set 0`, `set 1`, `set 2` et `set 3` respectivement. Vous trouverez ci-dessous les symboles que chacun de ces jeux utilise pour la numérotation. Notez que le nombre maximum qui peut être atteint est de 9 dans les jeux de symboles `set 0` et `set 1` et de 12 dans les jeux de symboles `set 2` et `set 3` :

```

Set 0: ● – ★ ▷ ◦ ○ □ ✓
Set 1: ★ ** *** ‡ ‡‡ ‡‡‡ * ** ***
Set 2: * † ‡ ** †† †‡ *** ††† †‡‡ **** †††† †‡‡‡
Set 3: ★ ** *** ‡ ‡‡ ‡‡‡ ¶ ¶¶ ¶¶¶ § §§ §§§

```

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 ».

ConT_EXt permet d'attribuer des configurations différentes aux commandes selon que l'on fait appel à elles dans telle où 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 nouveaux macro-section avec `\definesectionblock`.

Chapitre 8

Table des matières, index, listes

Table of Contents: **8.1 Table des matières;** 8.1.1 Vue d'ensemble de la table des matières; 8.1.2 Table des matières entièrement automatique avec un titre; 8.1.3 Table des matières automatique sans titre; 8.1.4 Option de sélection des éléments intégrés à la TdM : l'option criterium; 8.1.5 Mise en page de la TdM : l'option alternative; 8.1.6 Format des entrées de la TdM; 8.1.7 Ajustements manuels; A Inclure les sections non numérotées dans la TdM; B Ajouter des entrées manuellement à la TdM; C Exclure de la TdM une section particulière appartenant à un type de section qui est inclus dans le TdM; D Textes du titre différents entre TdM et corps du document; **8.2 Listes, listes combinées et tables des matières basées sur une liste;** 8.2.1 Listes de ConTeXt; 8.2.2 Listes ou index d'images, de tableaux et d'autres éléments; 8.2.3 Listes combinées; **8.3 Index;** 8.3.1 Génération de l'index; A The prior definition of the entries in the index and the marking of the points in the source file that refer to them; B Generating the final index; 8.3.2 Formatage de l'index; 8.3.3 Création d'autres index;

Une table des matières et un index sont des éléments globaux d'un document. Presque tous les documents auront une table des matières, tandis que seuls certains documents auront un index. Dans de nombreuses langues (mais pas en anglais), la table des matières et l'index sont regroupés sous le terme général «index». Pour les lecteurs anglais, la table des matières se trouve normalement au début (d'un document, voire dans certains cas au début des chapitres), et l'index à la fin.

L'un et l'autre impliquent une application particulière du mécanisme des références internes dont l'explication est incluse dans [section 9.2](#).

8.1 Table des matières

8.1.1 Vue d'ensemble de la table des matières

Dans le chapitre précédent, nous avons examiné les commandes qui permettent d'établir la structure d'un document au fur et à mesure de sa rédaction. Cette section se concentre sur la table des matières et l'index, qui reflètent en quelque sorte la structure du document et offrent des outils efficaces pour y naviguer. La table

des matières est très utile pour se faire une idée du document dans son ensemble (elle permet de contextualiser l'information) et pour rechercher l'endroit exact où se trouve un passage particulier. Les livres dont la structure est très complexe, avec de nombreuses sections et sous-sections de différents niveaux de profondeur, semblent nécessiter un autre type de table des matières, car une table des matières moins détaillée (peut-être avec seulement les deux ou trois premiers niveaux de sectionnement) est essentielle pour se faire une idée générale du contenu du document. Par contre, elle ne permet pas de localiser un passage particulier ce qui peut être l'objet d'une table des matières très détaillée. Cette dernière ayant pour faiblesse de facilement faire perdre la vue d'ensemble à son lecteur. C'est pourquoi, parfois, les livres dont la structure est particulièrement complexe comportent plusieurs tables des matières : une table des matières peu détaillée en début d'ouvrage, indiquant les parties principales, une table des matières plus détaillée au début de chaque chapitre ainsi que, éventuellement, un index en fin de document.

Tous ces éléments peuvent être générés automatiquement par ConTeXt avec une relative facilité. On peut :

- Générer une table des matières complète ou partielle à n'importe quel endroit du document.
- Décider du contenu de l'une ou l'autre.
- Configurer leur apparence jusqu'au moindre détail.
- Inclure des hyperliens qui nous permettent de passer directement à la section en question.

En fait, cette dernière fonctionnalité est incluse par défaut dans toutes les tables des matières, à condition que la fonction d'interactivité ait été activée dans le document. Voir, à ce sujet, [section 9.3](#).

L'explication de ce mécanisme dans le manuel de référence de ConTeXt est, à mon avis, quelque peu confuse, ce qui, je pense, est dû au fait que trop d'informations sont introduites en même temps. Le mécanisme de construction des tables des matières de ConTeXt comporte de nombreux éléments, et il est difficile de les expliquer tous tout en restant clair. En revanche, l'explication dans [wiki](#), se limite pratiquement des exemples : cela est très utile pour apprendre les *trucs* mais inadéquat – je pense – pour comprendre le mécanisme et comment il fonctionne. C'est pourquoi la stratégie que j'ai décidé d'utiliser pour expliquer les choses dans cette introduction commence par supposer quelque chose qui n'est pas strictement vrai (ou pas complètement vrai) : qu'il existe quelque chose dans ConTeXt appelé la *table des matières*. A partir de là, les commandes *normales* pour générer la table des matières sont expliquées, et quand ces commandes et leur configuration sont bien connues, je pense que c'est le moment d'introduire – bien qu'à un niveau théorique plutôt que plus pratique – les informations sur les pièces du mécanisme qui ont été omises jusqu'alors. La connaissance de ces *éléments supplémentaires* nous permet de créer des tables des matières beaucoup plus personnalisées que celles que nous pouvons appeler les *les éléments de base* créés avec les commandes expliquées jusqu'à ce point ; cependant, dans la plupart des cas, nous n'aurons pas besoin de le faire.

8.1.2 Table des matières entièrement automatique avec un titre

Les commandes de base pour générer automatiquement une table des matières (TdM ou ToC en anglais) à partir des sections numérotées d'un document (`part`, `chapter`, `section`, etc.) sont `\completecontent` et `\placecontent`. La principale différence entre ces deux commandes est que la première ajoute un *titre* à la Table des matières ; pour ce faire, elle insère immédiatement avant la Table des matières un *chapitre non numéroté* dont le titre par défaut est « Table des matières ».

Par conséquent, `\completecontent` :

- Insère, à l'endroit où il se trouve, un nouveau chapitre non numéroté intitulé « Table des matières ».

Nous rappelons que dans ConT_EXt la commande utilisée pour générer une section non numérotée au même niveau que les chapitres, est `\title`. (voir [section 7.2](#)). Par conséquent, en réalité, `\completecontent` n'insère pas un *Chapitre* (`\chapter`) mais un *Title* (`\title`). Je ne l'ai pas dit parce que je pense qu'il peut être déroutant pour le lecteur d'utiliser les noms des commandes de section non numérotées ici, puisque le terme *Title* a également un sens plus large, et il est facile pour le lecteur de ne pas l'identifier avec le niveau concret de section auquel nous faisons référence.

- Ce *chapitre* (en fait, `\title`) est formaté exactement de la même manière que le reste des chapitres non numérotés du document, ce qui inclut par défaut un saut de page.
- La table des matières est affichée immédiatement après le titre.

Initialement, la TdM générée est *complète*, comme nous pouvons le déduire du nom de la commande qui la génère (`\completecontent`). Mais d'une part, nous pouvons limiter le niveau de profondeur de la TdM comme expliqué dans [section 8.1.3](#) et, d'autre part, comme cette commande est *sensible* à l'endroit où elle se trouve dans le fichier source (voir ce qui est dit plus loin à propos de `\placecontent`), si `\completecontent` ne se trouve pas au début du document, il est possible que la TdM générée ne soit pas complète ; et à certains endroits du fichier source, il est même possible que la commande soit apparemment ignorée. Si cela se produit, la solution consiste à invoquer la commande avec l'option « `criterium=all` ». À propos de cette option, voir également [section 8.1.3](#).

Pour modifier le titre par défaut attribué aux TdM, nous utilisons la commande `\setupheadtext` dont la syntaxe est :

```
\setupheadtext[Langage][Élément=Texte]
```

où *Langage* est facultatif et fait référence à l'identificateur de langue utilisé par ConT_EXt (voir [section 10.5](#)), et *Élément* fait référence à l'élément dont nous voulons modifier le nom (« *content* » dans le cas de la table des matières) et *Texte* est le texte du titre que nous voulons donner à notre TdM. Par exemple

```
\setupheadtext[fr][content={Carte de navigation}]
```

fera en sorte que la TdM générée par `\completecontent` soit intitulée « Carte de navigation » au lieu de « Table des matières ».

De plus, `\completecontent` permet les mêmes options de configuration que `\placecontent`, pour l'explication desquelles je renvoie à la section suivante.

8.1.3 Table des matières automatique sans titre

La commande générale d'insertion d'une TdM sans titre, générée automatiquement à partir des commandes de sectionnement du document, est `\placecontent`, dont la syntaxe est :

```
\placecontent [Options]
```

En principe, la table des matières contiendra absolument toutes les sections numérotées, bien que nous puissions limiter son niveau de profondeur avec la commande `\setupcombinedlist` (dont nous parlerons plus loin). Ainsi, par exemple :

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

limitera le contenu de la table des matières aux deux niveaux indiqués : chapitre et section.

Une particularité de cette commande est qu'elle est sensible à son emplacement dans le fichier source. C'est très facile à expliquer avec quelques exemples, mais beaucoup plus difficile si nous voulons spécifier exactement comment la commande fonctionne et quelles rubriques sont incluses dans la Table des matières dans chaque cas. Commençons donc par les exemples :

- `\placecontent` placé au début du document, avant la première commande de section (partie, chapitre ou section, selon la situation) générera une table des matières complète.

Je ne suis pas vraiment sûr que la table des matières générée par défaut soit *complète*, je crois qu'elle comprend suffisamment de niveaux de section pour être complète dans la plupart des cas ; mais je soupçonne qu'elle ne dépassera pas le huitième niveau de section. Quoi qu'il en soit, comme mentionné ci-dessus, nous pouvons ajuster le niveau de sectionnement que la TdM atteint avec

```
\setupcombinedlist[content][list={chapitre, section, sous-section, ...}]
```

- En revanche, cette même commande située à l'intérieur d'une partie, d'un chapitre ou d'une section générera une TdM strictement limité au contenu de l'élément de section concerné, ou en d'autres termes des chapitres, des sections et d'autres niveaux inférieurs de découpage d'une partie spécifique, ou des sections (et d'autres niveaux) d'un chapitre spécifique, ou des sous-sections d'une section spécifique.

En ce qui concerne l'explication technique et détaillée, afin de bien comprendre le fonctionnement par défaut de `\placecontent`, il est essentiel de se rappeler que les différentes sections sont, en fait, des *environnements* pour ConTeXt Mark IV qui commencent par `\start\em TypedSection` et se terminent par `\stop\em TypedSection` et peuvent être contenues dans d'autres commandes de section de niveau inférieur. Donc, en tenant compte de cela, nous pouvons dire que `\placecontent` génère par défaut une table des matières qui comprendra uniquement :

- les éléments qui appartiennent à l'*environnement* (niveau de section) où la commande est placée. Cela signifie que la commande, lorsqu'elle est placée dans un chapitre, ne fera pas apparaître les sections et sous-sections des autres chapitres.
- les éléments qui ont un niveau de section inférieur au niveau correspondant au point où se trouve la commande. Cela signifie que si la commande se trouve dans un chapitre, seules les sections, les sous-sections et les autres niveaux inférieurs sont inclus ; Si la commande se trouve au niveau d'une section, seules les sous-sections, les sous-sous-sections et les autres niveaux inférieurs sont inclus.

En outre, pour que la table des matières soit générée, il faut que `\placecontent` se trouve *avant* la première section du chapitre dans lequel il se trouve, ou avant la première sous-section de la section dans laquelle il se trouve, etc.

Je ne suis pas sûr d'avoir été clair dans l'explication ci-dessus. Peut-être qu'avec un exemple un peu plus détaillé que les précédents, nous pourrions mieux comprendre ce que je veux dire : imaginons la structure suivante d'un document :

- Chapter 1
 - Section 1.1
 - Section 1.2
 - * Subsection 1.2.1
 - * Subsection 1.2.2
 - * Subsection 1.2.3
 - Section 1.3
 - Section 1.4
- Chapter 2

Ainsi : `\placecontent` placée avant le chapitre 1 générera une table des matières complète, similaire à celle générée par `\completecontent` mais sans titre. Mais si la commande est placée à l'intérieur du chapitre 1 et avant la section 1.1, la table des matières sera uniquement celle du chapitre ; et si elle est placée au début de la section 1.2, la table des matières sera uniquement le contenu de cette section. Mais si la commande est placée, par exemple, entre les sections 1.1 et 1.2, elle sera ignorée. Elle sera également ignorée si elle est placée à la fin d'une section, ou à la fin du document.

```

\setuphead[chapter][page=no]

\startchapter[title=chapter 1]

  \startcolor[darkgreen]
  \placecontent           % <=====
  \stopcolor

  \startsection[title=Section 1.1]
  \stopsection
  \startsection[title=Section 1.2]

    \startcolor[darkred]
    \placecontent         % <=====
    \stopcolor

    \startsubsection[title=Sous-section 1.2.1]
    \stopsubsection
    \startsubsection[title=Sous-section 1.2.2]
    \stopsubsection
    \startsubsection[title=Sous-section 1.2.3]
    \stopsubsection

  \stopsection
  \startsection[title=Section 1.3]
  \stopsection
  \startsection[title=Section 1.4]
  \stopsection

\stopchapter

\startchapter[title=chapter 2]
\stopchapter

```

1 chapter 1

1.1	Section 1.1	0
1.2	Section 1.2	0
1.2.1	Sous-section 1.2.1	0
1.2.2	Sous-section 1.2.2	0
1.2.3	Sous-section 1.2.3	0
1.3	Section 1.3	0
1.4	Section 1.4	0

1.1 Section 1.1

1.2 Section 1.2

1.2.1	Sous-section 1.2.1	0
1.2.2	Sous-section 1.2.2	0
1.2.3	Sous-section 1.2.3	0

1.2.1 Sous-section 1.2.1

1.2.2 Sous-section 1.2.2

1.2.3 Sous-section 1.2.3

1.3 Section 1.3

1.4 Section 1.4

2 chapter 2

Tout ceci, bien sûr, ne concerne que le cas où la commande ne comporte pas d'options. En particulier, l'option `criterium` modifiera ce comportement par défaut.

Parmi les options autorisées par `\placecontent` je n'en expliquerai que deux, les plus importantes pour la mise en place de la table des matières, et, de plus, les seules qui sont (partiellement) documentées dans le manuel de référence ConT_EXt. L'option `criterium`, qui affecte le contenu de la table des matières par rapport à l'endroit du fichier source où se trouve la commande, et l'option `alternative`, qui affecte la présentation générale de la table des matières à générer.

8.1.4 Option de sélection des éléments intégrés à la TdM : l'option `criterion`

Le fonctionnement par défaut de `\placecontent` vis-à-vis de la position de la commande dans le fichier source a été expliqué ci-dessus. En plus de la commande `\setuptupcombinedlist` vue à la [section A](#) qui permet de sélectionner les éléments à intégrer dans la TdM, l'option `criterion` permet d'adapter le fonctionnement. Entre autres, elle peut prendre les valeurs suivantes :

- item `all` : la TdM sera complète, quel que soit l'endroit du fichier source où se trouve la commande.
- `previous` : la table des matières n'inclura que les commandes de section (du niveau auquel nous nous trouvons) *précédent* la commande `\placecontent`. Cette option est destinée aux TdMs qui sont positionnées à en fin de document ou en fin de la section courante.
- `part`, `chapter`, `section`, `subsection`... : implique que la TdM doit être limitée au niveau de section indiqué.
- `component` : dans les projets multifichiers (voir [section 4.6](#)), cette option génère uniquement la TdM correspondant au *composant* (component) où se trouve la commande `\placecontent` ou `\completecontent`.

8.1.5 Mise en page de la TdM : l'option `alternative`

L'option `alternative` contrôle la présentation générale de la table des matières. Ses principales valeurs peuvent être consultées dans le [tableau 8.1](#).

alternative	Entrées sélectionnnées	Remarques
a	Numéro – Titre – Page	Une ligne par entrée
b	Numéro – Titre – Espaces – Page	Une ligne par entrée
c	Numéro – Titre – Ligne de points – Page	Une ligne par entrée
d	Numéro – Titre – Page	TdM continue, entrées bout-à-bout
e	Titre	Encadré
f	Titre	Justification à gauche
		Justification à droite ou centrée
g	Titre	Justification centrée

Tableau 8.1 Ways of formatting the table of contents

Les quatre premières alternatives (a, b, c, d) fournissent toutes les informations de chaque section (son numéro, son titre et le numéro de page où elle commence), et conviennent donc aussi bien aux documents papier qu'aux documents électroniques. Les trois dernières alternatives (e, f, g) ne nous informent que sur le titre, elles ne conviennent donc qu'aux documents électroniques où il n'est pas nécessaire de connaître le numéro de page où commence une section, à condition que la table des matières contienne un hyperlien vers celle-ci, ce qui est le cas par défaut avec ConT_EXt.

Par ailleurs, je pense que pour apprécier réellement les différences entre les différentes alternatives, il est préférable que le lecteur génère un document test où il pourra les analyser en détail. Voici quelques illustrations :

```
\setuphead[chapter][page=no]

\startcolor[darkgray]
\placecontent[alternative=a]      % <=====
\stopcolor

\startcolor[darkred]
\placecontent[alternative=b]      % <=====
\stopcolor

\startcolor[darkgreen]
\placecontent[alternative=c]      % <=====
\stopcolor

\startcolor[darkblue]
\placecontent[alternative=d]      % <=====
\stopcolor

\startcolor[darkcyan]
\placecontent[alternative=e]      % <=====
\stopcolor

\startcolor[darkmagenta]
\placecontent[alternative=f]      % <=====
\stopcolor

\startcolor[darkyellow]
\placecontent[alternative=g]      % <=====
\stopcolor

\startchapter[title=Chapitre 1]
\startsection[title=Section 1.1]
\startsubsection[title=Sous-section 1.1.1]
\stopsubsection
\stopsection
\stopchapter
```

```
1  Chapitre 1      0
1.1 Section 1.1    0
1.1.1 Sous-section 1.1.1  0

1  Chapitre 1      0
1.1 Section 1.1    0
1.1.1 Sous-section 1.1.1  0

1  Chapitre 1 ..... 0
1.1 Section 1.1 ..... 0
1.1.1 Sous-section 1.1.1 ..... 0
1 Chapitre 1 0 1.1 Section 1.1 0 1.1.1 Sous-section
1.1.1 0
Chapitre 1
Section 1.1
Sous-section 1.1.1

Chapitre 1
Section 1.1
Sous-section 1.1.1

1 Chapitre 1

1.1 Section 1.1

1.1.1 Sous-section 1.1.1
```

8.1.6 Format des entrées de la TdM

Nous avons vu que l'option alternative de `\placecontent` ou `\completecontent` nous permet de contrôler la *mise en page* générale de la table des matières, c'est-à-dire les informations qui seront affichées pour chaque rubrique, et la présence ou non de sauts de ligne séparant les différentes rubriques. Les ajustements finaux de chaque entrée de la table des matières sont effectués avec la commande `\setuplist` dont la syntaxe est la suivante :

```
\setuplist[Élément][Configuration]
```

où *Élément* fait référence à un type particulier de section. Il peut s'agir de *part*, *chapter*, *section*, etc. On peut également configurer plusieurs éléments en même temps, en les séparant par des virgules. *Configuration* offre jusqu'à 54 possibilités, dont beaucoup, comme d'habitude, ne sont pas expressément documentées ; mais cela n'empêche pas celles qui sont documentées, ou celles qui ne sont pas suffisamment claires, de permettre un ajustement très complet de la TdM. Je vais maintenant expliquer les options les plus importantes, en les regroupant selon leur utilité, mais avant de les aborder, rappelons qu'une entrée de la TdM, selon la valeur de l'option *alternative*, peut comporter jusqu'à trois éléments différents : Le numéro de section, le titre de la section et le numéro de page. Les options de configuration nous permettent de configurer les différents composants de manière globale ou séparée :

- **Inclusion ou exclusion des différents composants** : Si nous avons choisi une alternative qui inclut, en plus du titre, le numéro de section et le numéro de page (alternatives «a» «b» «c» ou «d»), les options `headnumber=no` ou `pagenumber=no` indiquent pour le niveau spécifique que nous configurons, que le numéro de section (`headnumber`) ou le numéro de page (`pagenumber`) ne doivent pas être affichés.
- **Couleur et style** : Nous savons déjà que l'entrée qui génère une section spécifique dans la TdM peut avoir (selon l'alternative) jusqu'à trois composants différents : le numéro de section, le titre et le numéro de page. Nous pouvons indiquer conjointement le style et la couleur des trois composants à l'aide des options `style` et `couleur`, ou le faire individuellement pour chaque composant au moyen des options `numberstyle`, `textstyle` et `pagestyle` (pour le style) et `numbercolor`, `textcolor` ou `pagecolor` pour la couleur.

Pour contrôler l'apparence de chaque entrée, en plus du style lui-même, nous pouvons appliquer une commande à l'entrée entière ou à l'un de ses différents éléments. Pour cela, il existe les options `command`, `numbercommand`, `pagecommand` et `textcommand`. La commande indiquée ici peut être une commande standard ConT_EXt ou une commande de notre propre création. Le numéro de section, le texte du titre et le numéro de page seront passés comme arguments à l'option `command`, tandis que le titre de la section sera passé comme argument à `textcommand` et le numéro de page à `pagecommand`. Ainsi, par exemple, la phrase suivante fera en sorte que les titres de section soient écrits en (fausses) petites majuscules :

```

\setuphead[chapter][page=no]

\setuplist
[section]
[textcommand=\cap,
pagecolor=darkgreen,
numberstyle=bold]

\placecontent[alternative=c] % <=====

\startchapter[title=Chapitre 1]
\startsection[title=Section 1.1]
\startsubsection[title=Sous-section 1.1.1]
\stopsubsection
\stopsection
\stopchapter

```

1	Chapitre 1	0
1.1	SECTION 1.1	0
1.1.1	Sous-section 1.1.1	0

1 Chapitre 1

1.1 Section 1.1

1.1.1 Sous-section 1.1.1

- **Séparation des autres éléments de la TdM** : Les options `before` et `after` nous permettent d'indiquer les commandes qui seront exécutées avant (`before`) et après (`after`) la composition de l'entrée de la table des matières. Normalement, ces commandes sont utilisées pour définir l'espacement ou un élément de séparation (tel un trait horizontal) entre les entrées précédentes et suivantes.
- **Indentation d'un élément** : défini avec l'option `margin` qui nous permet de définir l'espace d'indentation gauche des entrées du niveau que nous configurons.
- **Hyperliens intégrés à la TdM** : Par défaut, les entrées de l'index comprennent un hyperlien vers la page du document où commence la section en question. L'option `interaction` permet de désactiver cette fonction (`interaction=no`) ou de limiter la partie de l'entrée d'index où se trouvera l'hyperlien, qui peut être le numéro de section (`interaction=number` ou `interaction=section-number`), le titre de la section (`interaction=text` ou `interaction=title`) ou le numéro de page (`interaction=page` ou `interaction=pagenumber`).
- *Autres aspects*:
 - `width` et `distance` : spécifient respectivement la largeur accordée à l'affichage du numéro et la distance de séparation entre l'espace d'affichage du numéro et le titre de la section. Il peut s'agir d'une dimension (ou du mot clé `fit` pour `width` qui définit la largeur exacte du numéro de section).
 - `numeralalign` : indique l'alignement des éléments de numérotation ; il peut être `left`, `right`, `middle`, `flushright`, `flushleft`, `inner`, `outer`

Voici un exemple qui combine ces trois dernières options :


```

\setuphead[chapter] [page=no]

\setuplist[chapter] [width=15mm, margin=0mm,
                    distance=3mm,
                    numberalign=flushright]
\setuplist[section] [width=15mm, margin=0mm,
                    distance=6mm,
                    numberalign=flushright]
\setuplist[subsection] [width=15mm, margin=0mm,
                      distance=9mm,
                      numberalign=flushright]

\placecontent

\startchapter [title=Chapitre 1]
\startsection [title=Section 1.1]
  \startsubsection [title=Sous-section 1.1.1]
  \stopsubsection
\stopsection
\stopchapter

```

1	Chapitre 1	0
1.1	Section 1.1	0
1.1.1	Sous-section 1.1.1	0

1 Chapitre 1

1.1 Section 1.1

1.1.1 Sous-section 1.1.1

- `symbol` : permet de remplacer le numéro de section par un *symbole*. Trois valeurs possibles sont prises en charge : `one`, `two` et `three`. La valeur `none` de cette option supprime le numéro de section de la table des matières.

Parmi les multiples options de configuration de la TdM, aucune ne nous permet de contrôler directement l'espacement entre les lignes. Celui-ci sera, par défaut, celui qui s'applique à l'ensemble du document. Souvent, cependant, il est préférable que les lignes de la table des matières soient légèrement plus serrées que le reste du document. Pour ce faire, nous devons inclure la commande qui génère la table des matières (`\placecontent` ou `\completecontent`) dans un groupe où l'espacement interligne est défini différemment. Par exemple :

```

\setuphead[chapter] [page=no]
\start
  \startcolor [darkred]
  \setupinterlinespace [6mm]
  \placecontent
  \stopcolor
\stop

\start
  \startcolor [darkyellow]
  \setupinterlinespace [small]
  \placecontent
  \stopcolor
\stop

\startchapter [title=Chapitre 1]
\startsection [title=Section 1.1]
  \startsubsection [title=Sous-section 1.1.1]
  \stopsubsection
\stopsection
\stopchapter

```

1	Chapitre 1	0
1.1	Section 1.1	0
1.1.1	Sous-section 1.1.1	0

1	Chapitre 1	0
1.1	Section 1.1	0
1.1.1	Sous-section 1.1.1	0

1 Chapitre 1

1.1 Section 1.1

1.1.1 Sous-section 1.1.1

8.1.7 Ajustements manuels

Nous avons déjà expliqué les deux commandes fondamentales pour générer des tables des matières (`\placecontent` et `\completecontent`), ainsi que leurs options. Ces deux commandes permettent de générer automatiquement des tables des matières, construites à partir des sections numérotées existantes dans le document, ou dans le bloc ou le segment du document auquel la table des matières fait référence. Je vais maintenant expliquer certaines configurations que nous pouvons effectuer pour que le contenu de la table des matières ne soit pas aussi *automatique* et plus *personnalisé*. Cela implique :

- la possibilité d’inclure également certains titres de sections non numérotées dans la table des matières.
- la possibilité d’intégrer dans la table des matières une entrée particulière qui ne correspond pas à une section numérotée.
- la possibilité d’exclure une section numérotée particulière de la table des matières.
- la possibilité que le titre d’une section particulière figurant dans la table des matières ne coïncide pas exactement avec le titre figurant dans le corps du document.

A. Inclure les sections non numérotées dans la TdM

Le mécanisme par lequel ConTeXt construit la TdM implique que toutes les sections numérotées sont automatiquement incluses, ce qui, comme je l’ai déjà dit (voir [section 7.4.2](#)) dépend des deux options (`number` et `incrementnumber`) que nous pouvons modifier avec `\setuphead` pour chaque type de section. Il a également été expliqué qu’un type de section où `incrementnumber=yes` et `number=no` serait une section numérotée *en interne* mais pas *en externe* (c’est à dire pas de façon visible).

Par conséquent, si nous voulons qu’un type de section non numéroté particulier – par exemple, `subject` – soit inclus dans la table des matières, nous devons modifier la valeur de l’option `incrementnumber` pour ce type de section, en lui attribuant la valeur `yes`, puis inclure ce type de section parmi ceux qui doivent être affichés dans la table des matières, ce qui se fait, comme expliqué précédemment, avec `\setupcombinedlist`.

Nous pouvons ensuite, si nous le souhaitons, formater cette entrée en utilisant `\setuptuplist` de la même manière que les autres ; par exemple :

```

\setuphead [chapter] [page=no]

\setuphead [subject]
    [incrementnumber=yes]
\setuplist [subject] [color=darkred]

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

\placecontent

\startchapter [title=Chapitre 1]
\startsection [title=Section 1.1]
\stopsection
\startsubject [title=Sujet 1.2]
\stopsubject
\stopchapter

```

1	Chapitre 1	0
1.1	Section 1.1	0
1.2	Sujet 1.2	0

1 Chapitre 1

1.1 Section 1.1

1.2 Sujet 1.2

Note: La procédure qui vient d'être expliquée inclura toutes les occurrences dans notre document du type de section non numérotée concerné (dans notre exemple, les sections de type subject). Si l'on ne souhaite inclure qu'une occurrence particulière de ce type de section dans la table des matières, il est préférable de le faire par la procédure expliquée ci-dessous.

B. Ajouter des entrées manuellement à la TdM

On peut envoyer soit une entrée (simulant l'existence d'une section qui n'existe pas réellement), soit une commande à la table des matières, à partir de n'importe quel point du fichier source.

Pour envoyer une entrée qui simule l'existence d'une section qui n'existe pas réellement, utilisez la fonction `\writetolist` dont la syntaxe est :

```

\writetolist [TypedSection] [Options] {Numéro}{Texte}

```

dans laquelle :

- Le premier argument indique le niveau que doit avoir cette entrée de section dans la table des matières : `chapter`, `section`, `sous-section`, etc.
- Le deuxième argument, qui est facultatif, permet de configurer cette entrée d'une manière particulière. Si l'entrée envoyée manuellement est omise, elle sera formatée comme le sont toutes les entrées du niveau indiqué par le premier argument ; je dois cependant signaler que dans mes tests, je n'ai pas réussi à le faire fonctionner.

Tant dans la liste officielle des commandes de ConTExT (voir section ??) que dans le wiki, on nous dit que cet argument permet les mêmes valeurs que `\setuplist` qui est la commande qui nous permet de formater les différentes entrées de la COT. Mais, j'insiste, dans mes tests je n'ai pas réussi à modifier de quelque manière que ce soit l'apparence de l'entrée de la TdM envoyée manuellement.





- Le troisième argument est censé refléter la numérotation que possède l'élément envoyé à la COT, mais je n'ai pas non plus réussi à le faire fonctionner dans mes tests.
- Le dernier argument comprend le texte à envoyer à la table des matières.

Ceci est utile, par exemple, si nous voulons envoyer une section non numérotée particulière, mais seulement celle-ci à la table des matières. Dans [section A](#), il est expliqué comment obtenir qu'une catégorie entière de sections non numérotées soit envoyée à la table des matières ; mais si nous voulons seulement y envoyer une occurrence particulière d'un type de section, il est plus pratique d'utiliser la commande `\writetolist` localement. Ainsi, par exemple, si nous voulons que la section de notre document contenant la bibliographie ne soit pas une section numérotée, mais qu'elle soit tout de même incluse dans la table des matières, nous devons écrire :

```
\setuphead [chapter] [page=no]
\placecontent
\startchapter [title=Chapitre 1]
\startsection [title=Section 1.1]
\stopsection
\startsubject [title=Sujet]
\writetolist [section]{53}{Sujet important}
\stopsubject
\stopchapter
```

1	Chapitre 1	0
1.1	Section 1.1	0
	Sujet important	0

1 Chapitre 1

1.1 Section 1.1

Sujet

Voyez comment nous utilisons la version non numérotée de `section`, qui est `subject`, pour la section mais nous l'envoyons à l'index, manuellement, comme s'il s'agissait d'une section numérotée (`section`).

Une autre commande destinée à influencer manuellement la table des matières est `\writebetweenlist` qui est utilisée pour envoyer non pas une entrée elle-même, mais une *commande* à la table des matières, depuis un point particulier du document. Par exemple, si nous voulons inclure une ligne entre deux éléments de la TdM, nous pouvons écrire ce qui suit à n'importe quel endroit du document situé entre les deux sections concernées :

```

\setuphead [chapter] [page=no]

\placecontent

\startchapter [title=Chapitre 1]
\startsection [title=Section 1.1]
\stopsection
\writebetweenlist
  [section]
  [location=here]
  {\hrule}
\startsection [title=Section 1.2]
\stopsection
\stopchapter

```

1	Chapitre 1	0
1.1	Section 1.1	0
1.2	Section 1.2	0

1 Chapitre 1

1.1 Section 1.1

1.2 Section 1.2

C. Exclure de la TdM une section particulière appartenant à un type de section qui est inclus dans le TdM

La table des matières est construite à partir de *types de section* établis, comme nous le savons déjà, par l'option `list` de `\setupcombinedlist`. Par conséquent, si un certain *type de section* doit apparaître dans la table des matières, il n'y a aucun moyen d'en exclure une section particulière que nous ne voulons pas voir figurer dans la table des matières, pour quelque raison que ce soit.

Normalement, si l'on ne veut pas qu'une section apparaisse dans la table des matières, il faut utiliser son équivalent non numéroté, c'est-à-dire, par exemple, `title` au lieu de `chapter`, `subject` au lieu de `section`, etc. Ces sections ne sont pas envoyées à la TdM, et ne sont pas non plus numérotées.

Cependant, si pour une raison quelconque, nous voulons qu'une certaine section soit numérotée mais n'apparaisse pas dans la table des matières, même si d'autres types de ce genre le font, nous pouvons utiliser une *astuce* qui consiste à créer un nouveau type de section qui est un clone de la section en question. Par exemple :

```

\definehead [MySubsection] [subsection]
\placecontent
\startsection [title={Première section}]
\stopsection
\startsubsection [title={Première subsection}]
\stopsubsection
\startMySubsection [title={Seconde subsection}]
\stopMySubsection
\startsubsection [title={Troisième subsection}]
\stopsubsection

```

1	Première section	0
1.1	Première subsection	0
1.3	Troisième subsection	0

1 Première section

1.1 Première subsection

1.2 Seconde subsection

1.3 Troisième subsection

Ainsi, lors de l'insertion d'une section de type `MySubsection`, le compteur de sous-sections augmentera, puisque cette section est un *clone* des sous-sections, mais la TdM ne sera pas modifiée, puisque par défaut elle n'inclut pas les section de types `MySubsection`.

D. Textes du titre différents entre TdM et corps du document

Si nous voulons un texte de titre d'une section particulière différent entre celui inclu dans la table des matières et celui affiché dans le corps du document, nous devons créer la section non pas avec la syntaxe traditionnelle (`SectionType{Titre}`) mais avec `\SectionType [Options]`, ou avec `\startSectionType [Options]`, et attribuer le texte que l'on souhaite voir écrit dans la Table des matières à l'option `list` (voir section ??).

```
\placecontent
\startsection
  [title={Une introduction approximative
et légèrement répétitive à la réalité de
l'évidence},
  list={Une introduction à la réalité de
l'évidence}]
\stopsection
```

1 Une introduction à la réalité de l'évidence 0

1 Une introduction approximative et légèrement
répétitive à la réalité de l'évidence

8.2 Listes, listes combinées et tables des matières basées sur une liste

En interne, pour ConT_EXt une table des matières n'est rien de plus qu'une *liste combinée*, qui, à son tour, comme son nom l'indique, consiste en une combinaison de listes simples. Par conséquent, la notion de base à partir de laquelle ConT_EXt construit la table des matières est celle d'une liste. Plusieurs listes sont combinées pour former une table des matières. Par défaut, ConT_EXt contient une liste combinée prédéfinie appelée « content » et c'est avec elle que les commandes examinées jusqu'à présent fonctionnent : `\placecontent` et `\completecontent`.

8.2.1 Listes de ConT_EXt

Dans ConT_EXt, une *liste* est plus précisément une liste d'éléments numérotés à propos desquels nous devons nous souhaitons conserver trois informations :

1. leur numéro.
2. leur texte, qui peut aussi être leur nom, leur titre.
3. leur page d'apparition.

Cela est utilisé avec les sections numérotées, mais aussi avec d'autres éléments du document tels que les images, les tableaux, etc. En général, les éléments pour lesquels il existe une commande dont le nom commence par `\place` qui les positionne dans le document comme `\placetable`, `\placefigure`, etc.

Dans tous ces cas, ConT_EXt génère automatiquement une liste des différentes occurrences du type d'élément en question, avec pour chaque occurrence : son numéro, son titre (ou texte) et sa page. Ainsi, par exemple, il existe une liste de chapitres, appelée `chapter`, une autre de sections, appelée `section` ; mais aussi une autre de tableaux (appelée `table`) ou d'images (appelée `figure`). Les listes générées automatiquement par ConT_EXt sont toujours appelées de la même manière que l'élément qu'elles stockent.

Une liste sera également générée automatiquement si nous créons, par exemple, un nouveau type de section numérotée : lorsque nous la créons, nous créons aussi implicitement la liste qui les stocke. Et si, pour une section non numérotée par défaut, nous activons l'option `incrementnumber=yes`, ce qui en fait une section numérotée, nous créerons aussi implicitement une liste portant ce nom.

Outre les listes implicites (définies automatiquement par ConT_EXt), nous pouvons créer nos propres listes avec `\definelist`, dont la syntaxe est la suivante

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

Des éléments sont ajoutés de la liste :

- Dans les listes prédéfinies par ConT_EXt, ou créées par lui suite à la création d'un nouvel objet flottant (voir section ??), automatiquement chaque fois qu'un élément de la liste est inséré dans le document, soit par une commande de section,

soit par la commande `\placeQuoiQueCeSoit` pour les autres types de listes, par exemple : `\placefigure`, insère une image dans le document, mais elle insère également l'entrée correspondante dans la liste.

- Manuellement dans tout type de liste avec `\writetolist[ListName]`, déjà expliqué dans [subsection B](#) de [section 8.1.7](#). La commande `\writebetweenlist` est également disponible. Elle a également été expliquée dans cette section.

Une fois qu'une liste a été créée et que tous ses éléments y ont été inclus, les trois commandes de base qui s'y rapportent sont `\setuplist`, `\placelist` et `\completetelist`. La première nous permet de configurer l'aspect de la liste ; les deux dernières insèrent la liste en question à l'endroit du document où elle les trouve. La différence entre `\placelist` et `\completetelist` est similaire à la différence entre `\placecontent` et `\completecontent` : `complete` introduit la liste dans une section dédiée, avec un titre dédié, de niveau `chapter / title`. (voir les [sections 8.1.2](#) et [8.1.3](#)).

Donc, par exemple `\placelist[section]` insère une liste des sections, y compris un lien hypertexte vers celles-ci (si l'interactivité du document est activée et si, dans `\setuplist`, nous n'avons pas défini `interaction=no`). Une liste de sections n'est pas exactement la même chose qu'une table des matières basée sur des sections : l'idée d'une table des matières inclut généralement aussi les niveaux supérieurs et inférieurs (sous-sections, sous-sous-sections, etc.). Mais une liste de sections ne comprendra que les sections elles-mêmes.

```
\placelist[section]

\startsection[title=Section 1.1]
\stopsection

\startsection[title=Section 1.2]
  \startsubsection[title=Sous-section 1.2.1]
  \stopsubsection
  \startsubsection[title=Sous-section 1.2.2]
  \stopsubsection
  \startsubsection[title=Sous-section 1.2.3]
  \stopsubsection
\stopsection

\startsection[title=Section 1.3]
\stopsection
\startsection[title=Section 1.4]

\stopsection
```

1	Section 1.1	0
2	Section 1.2	0
3	Section 1.3	0
4	Section 1.4	0

1	Section 1.1
2	Section 1.2
2.1	Sous-section 1.2.1
2.2	Sous-section 1.2.2
2.3	Sous-section 1.2.3
3	Section 1.3
4	Section 1.4

La syntaxe de ces commandes est la suivante :

```
\placelist[ListName] [Options]}  
\setuplist[ListName] [Configuration]
```

Les options de `\setuplist` ont déjà été expliquées dans [section 8.1.6](#), et les options de `\placelist` sont les mêmes que celles de `\placecontent` (voir [section 8.1.3](#)).

8.2.2 Listes ou index d’images, de tableaux et d’autres éléments

D’après ce qui a été dit jusqu’à présent, on peut voir que, puisque ConTeXt crée automatiquement une liste d’images placées dans un document avec la commande `\placefigure`, générer une liste ou un index d’images à un point particulier de notre document est aussi simple que d’utiliser la commande `\placelist[figure]`. Et si nous voulons générer une liste avec un titre (similaire à ce que nous obtenons avec `\completecontent`), nous pouvons le faire avec `\completelist[figure]`. Nous pouvons faire de même avec les quatre autres types prédéfinis d’objets flottants dans ConTeXt : tableaux (« table »), graphiques (« graphic »), *intermezzi* (« intermezzo ») et formules chimiques (« chemical »). ConTeXt comprend également des commandes qui les génère dans leur version sans titre (`\placelistoffigures`, `\placelistoftables`, `\placelistofgraphics`, `\placelistofintermezzi` et `\placelistofchemicals`), et d’autre qui les génère avec titre (`\completelistoffigures`, `\completelistoftables`, `\completelistofgraphics`, `\completelistofintermezzi` et `\completelistofchemicals`), de manière similaire à `\completecontent`.

De la même manière, pour les objets flottants que nous avons nous-mêmes créés (voir [section ??](#)), les commandes `\placelistof<FloatName>` et `\completelistof<FloatName>` seront automatiquement créés.

Pour les listes que nous avons créées avec `\definelist[ListName]`, nous pouvons créer un index avec `\placelist[ListName]` ou avec `\completelist[ListName]`.

8.2.3 Listes combinées

Une liste combinée est, comme son nom l’indique, une liste qui combine des éléments provenant de différentes listes précédemment définies. Par défaut, ConTeXt définit une liste combinée pour les tables de contenu dont le nom est « content », mais nous pouvons créer d’autres listes combinées avec `\definecombinedlist` dont la syntaxe est :

```
\definecombinedlist[Nom] [Listes] [Options]
```

où :

- *Nom* : est le nom que portera la nouvelle liste combinée.
- *Listes* : désigne les noms des listes à combiner, séparés par des virgules.
- *Options* : Options de configuration de la liste. Elles peuvent être indiquées au moment de la définition de la liste, ou, de préférence, lorsque la liste est invoquée. Les principales options (qui ont déjà été expliquées) sont les suivantes : *criterium* (sous-section 8.1.4 de section 8.1.3) et *alternative* (dans sous-section ?? de la même section).

Un effet collatéral de la création d’une liste combinée avec `\definecombinedlist` est qu’elle crée également une commande appelée `\placeListNom` qui sert à invoquer la liste, c’est-à-dire à l’inclure dans le fichier de sortie. Ainsi, par exemple,

```
\definecombinedlist [TdM]
\definecombinedlist [content]
```

créera les commandes `\placeTdM` et `\placecontent`.

Mais attendez, `\placecontent` ! N’est-ce pas la commande qui est utilisée pour créer une table des matières *normal* ? En effet, cela signifie que la table des matières standard est en fait créée par ConT_EXt au moyen de la commande suivante :

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

Une fois notre liste combinée définie, nous pouvons la configurer (ou la reconfigurer) avec `\setupcombinedlist` qui permet les options déjà expliquées *criterium* (voir sous-section 8.1.4 dans section 8.1.3) et *alternative* (voir subsection 8.1.5 dans la même section), ainsi que l’option *list* pour *modifier* les listes incluses dans la liste combinée.

La liste officielle des commandes ConT_EXt (voir section ??) ne mentionne pas l’option *list* parmi les options autorisées pour `\setupcombinedlist`, mais elle est utilisée dans plusieurs exemples d’utilisation de cette commande dans le wiki (qui, par ailleurs, ne la mentionne pas non plus dans la page consacrée à cette commande sauf en exemple). J’ai également vérifié que l’option fonctionne.

8.3 Index

8.3.1 Génération de l'index

Générer un index consiste à générer la liste des sujets considérés comme importants ou significatifs et précisant les pages y faisant référence. L'index est généralement située à la fin d'un document.

Lorsque les livres étaient composés à la main, la création d'un index des sujets était une tâche complexe et fastidieuse. Tout changement dans la pagination pouvait affecter toutes les entrées de l'index. Par conséquent, ils n'étaient pas très courants. Aujourd'hui, les mécanismes informatiques de composition font que, même si la tâche est susceptible de rester fastidieuse, elle n'est plus aussi complexe étant donné qu'il n'est pas si difficile pour un système informatique de maintenir à jour la liste des données associées à une entrée d'index.

Pour générer un index par sujet, nous avons besoin de :

1. Déterminer les mots, les termes ou les concepts qui doivent faire partie de l'article. Il s'agit d'une tâche que seul l'auteur peut accomplir.
2. Vérifier à quels endroits du document chaque entrée du futur index apparaît. Bien que, pour être précis, plus que *vérifier* les endroits du fichier source où le concept ou la question est abordé, ce que nous faisons lorsque nous travaillons avec ConTeXt consiste à *marquer* ces endroits, en insérant une commande qui servira ensuite à générer l'index automatiquement. C'est la partie la plus fastidieuse.
3. Enfin, nous générons et mettons en forme l'index en le plaçant à l'endroit de notre choix dans le document. Cette dernière opération est assez simple avec ConTeXt et ne nécessite qu'une seule commande : `\placeindex` ou `\completeindex`.

A. The prior definition of the entries in the index and the marking of the points in the source file that refer to them

The fundamental work is in the second step. It is true that computer systems also facilitate it in the sense that we can do a global text search to locate the places in the source file where a specific subject is treated. But we should also not blindly rely on such text searches: a good subject index must be able to detect every spot where a particular subject is being discussed, even if this is done without using the *standard* term to refer to it.

To *mark* an actual point in the source file, associating it with a word, term or idea that will appear in the index, we use the `\index` command whose syntax is as follows:

```
\index[Alphabetical][Index entry]
```

where *Alphabetical* is an optional argument that is used to indicate an alternative text to that of the index entry itself in order to sort it alphabetically, and *Index entry* is the text that will appear in the index, associated with this mark. We can also apply

the formatting features that we wish to use, and if reserved characters appear in the text, they must be written in the usual way in 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{\backslash TeX}` will list the command not by the «t» in «TeX», but among the symbols, since the term sent to the index begins with a backslash. This is done by writing `\index[tex]{\backslash TeX}`.

The *index entries* will be the ones we want. For a subject index to be really useful we have to work a little harder at asking what concepts the reader of a document is most likely to look for; so, for example, it may be better to define an entry as « disease, Hodgkins » than defining it as « Hodgkin’s disease », since the more inclusive term is « disease ».

By convention, entries in a subject index are always written in lower case, unless they are proper names.

If the index has several levels of depth (up to three are allowed) to associate a particular index entry with a specific level the «+» character is used. As follows:

```
\index{Entry 1+Entry 2}  
\index{Entry 1+Entry 2+Entry 3}
```

In the first case we defined a second level entry called *Entry 2* that will be a sub-entry of *Entry 1*. In the second case we defined a third level entry called *Entry 3* that will be a sub-entry of *Entry 2*, which in turn is a sub-entry of *Entry 1*. For example

```
My \index{dog}dog, is a \index{dog+greyhound}greyhound called Rocket.  
He does not like \index{cat+stray}stray cats.
```

It is worth noting some details of the above:

- The `\index` command is usually placed *before* the word it is associated with and is normally not separated from it by a blank space. This is to ensure that the command is on the exact same page as the word it is linked to:
 - If there were a space separating them, there could be the possibility that ConT_EXt 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 Formatage de l'index

Les index de sujets sont une application particulière d'une structure plus générale que ConTeXt appelle « *register* » ; par conséquent l'index est formaté avec la commande :

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

Avec cette commande, nous pouvons :

- Déterminer à quoi ressemblera l'index avec ses différents éléments. A savoir :
 - Les titres de l'index qui sont généralement des lettres de l'alphabet. Par défaut, ils sont en minuscules. Avec `alternative=A`, nous pouvons les mettre en majuscules.
 - Les entrées elles-mêmes, et leur numéro de page. L'apparence dépend des options `textstyle`, `textcolor`, `textcommand` et `deeptextcommand` pour l'entrée elle-même, et `pagestyle`, `pagecolor` et `pagecommand`, pour le numéro de page. Avec `pagenumber=no`, on peut aussi générer un index des sujets sans numéro de page (mais je ne sais pas si cela peut être utile).
 - L'option `distance` spécifie la largeur de séparation entre le nom d'une entrée et les numéros de page ; mais elle mesure aussi la quantité d'indentation pour les sous-entrées.

Les noms des options `style`, `textstyle`, `pagestyle`, `color`, `textcolor`, et `pagecolor` sont suffisamment clairs pour nous dire ce que chacune fait, je pense. Pour les options `command`, `pagecommand`, `textcommand` et `deeptextcommand`, je me réfère à l'explication des options de même nom dans [section 7.4.4](#), concernant la configuration des commandes de section.

- Pour définir l'apparence générale de l'index, ce qui inclut, entre autres, les commandes à exécuter avant (`before`) ou après (`after`) l'index, le nombre de colonnes qu'il doit avoir (`n`), si les colonnes doivent être d'égale hauteur ou non (`balance`), l'alignement des entrées (`align`), etc.

8.3.3 Création d'autres index

J'ai expliqué l'index des sujets comme si un seul index de ce type était possible dans un document ; mais la vérité est que les documents peuvent avoir autant d'index que souhaité. Il peut y avoir un index des noms de personnes, par exemple, qui rassemble les noms des personnes mentionnées dans le document, avec une indication de la page où elles sont citées. Il s'agit toujours d'une sorte d'index. Dans un texte juridique, nous pourrions également créer un index spécial pour les mentions du Code civil ; ou, dans un document comme le présent, un index des macros expliquées dans celui-ci, etc.

Pour créer un index supplémentaire dans notre document, nous utilisons la commande `\defineregister` dont la syntaxe est :

```
\defineregister [NomIndex] [Configuration]
```

où *NomIndex* est le nom que portera le nouvel index, et *Configuration* contrôle son fonctionnement. Il est également possible de configurer l'index ultérieurement au moyen de la fonction

```
\setupregister [NomIndex] [Configuration]
```

Une fois qu'un nouvel index nommé *NomIndex* a été créé, nous disposons de la commande `\NomIndex` pour marquer les entrées de cet index de la même manière que les entrées sont marquées avec `\index`. La commande `\seeNomIndex` nous permet également de créer des entrées qui font référence à d'autres entrées.

Par exemple, nous pouvons créer un index des commandes ConTeXt dans ce document avec la commande :

```
\defineregister[MaMacro]

La commande \tex{etbim} permet de...
\MaMacro{etbim}

\startsubject[title=Index des macros]
\placeMaMacro
\stopsubject
```

La commande `\etbim` permet de...

Index des macros

e
etbim 0

qui crée la commande `\MaMacro`. Cela me permet de marquer toutes les références aux commandes ConTeXt comme une entrée d'index, puis de générer l'index avec `\placeMaMacro` ou `\completeMaMacro`.

La création d'un nouvel index active la commande `\NomIndex` pour le marquer comme entrée, et les commandes `\placeNomIndex` et `\completeNomIndex` pour générer l'index. Mais ces deux dernières commandes sont en fait des abréviations de deux commandes plus générales appliquées à l'index en question. Ainsi, `\placeNomIndex` est équivalent à `\placeregister [NomIndex]` et `\completeNomIndex` est équivalent à `\completeegister [NomIndex]`.

Chapitre 9

Références et hyperliens

Table of Contents: 9.1 Types de référence; 9.2 Références internes; 9.2.1 L'étiquette de la cible; 9.2.2 Commandes au point de référence d'origine pour récupérer les données du point cible; A Commandes de base pour récupérer les informations d'une étiquette; B Récupération des informations associées à une étiquette avec la commande `\ref`; C Détecter où le lien mène; 9.2.3 Génération automatique de préfixes pour éviter les étiquettes en double; 9.3 Documents électroniques interactifs; 9.3.1 Permettre l'interactivité dans les documents; 9.3.2 Configuration de base pour les éléments interactifs; 9.4 Hyperliens vers des documents externes; 9.4.1 Commandes de définition d'hyperlien (sans les introduire dans le document); 9.4.2 Commandes d'insertion d'hyperlien dans le document; 9.5 Création de signets dans le PDF final; 9.6 Pièces jointes;

9.1 Types de référence

Les documents scientifiques et techniques abondent de références :

- Parfois, ils se réfèrent à d'autres documents qui sont à la base de ce qui est expliqué, ou qui contredisent ce qui est expliqué, ou qui développent ou nuancent l'idée traitée, etc. Dans ce cas, la référence est dite *externe* et, si l'on veut que le document soit rigoureux sur le plan académique, la référence prend la forme de *citations* de la littérature.
- Mais il est également fréquent qu'un document, dans l'une de ses sections, fasse référence à une autre de ses sections, auquel cas la référence est dite *interne*. Lorsqu'un point du document commente un aspect particulier d'une image, d'un tableau, d'une note ou d'un élément de même nature, en s'y référant par son numéro ou par la page où il se trouve, il s'agit également d'une référence *interne*.

Pour des raisons de précision, les références internes doivent viser un endroit précis et facilement identifiable dans le document. C'est pourquoi ce type de références renvoie toujours soit à des éléments numérotés (comme, par exemple, lorsque nous disons « voir tableau 3.2 », ou « chapitre 7 »), soit à des numéros de page. Les références vagues du type « comme nous l'avons déjà dit » ou « comme nous le verrons plus loin » ne sont pas de véritables références, et il n'y a pas d'exigence particulière pour les mettre en forme, ni d'outil spécial pour le faire.

De plus, je dissuade personnellement mes étudiants de doctorat ou de maîtrise d'utiliser cette pratique de manière habituelle.

Les références internes sont aussi communément appelées « références croisées » bien que dans ce document j'utiliserai simplement le terme « références » en général, et « références internes » lorsque je souhaite être spécifique.

Afin de clarifier la terminologie que j'utilise pour les références, j'appellerai le point du document où une référence est introduite, l'« origine », et l'endroit vers lequel elle pointe, la *quotationcible*. De cette façon, nous dirons qu'une référence est interne lorsque l'origine et la cible sont dans le même document, et externe lorsque l'origine et la cible sont dans des documents différents.

Du point de vue de la composition du document :

- Les références externes ne posent aucun problème particulier et ne nécessitent donc, en principe, aucun outil pour les introduire : toutes les données dont j'ai besoin à partir du document cible sont à ma disposition et je peux les utiliser dans la référence. Toutefois, si le document d'origine est un document électronique et que le document cible est également disponible sur le Web, il est possible d'inclure dans la référence un lien hypertexte permettant de passer directement à la cible par un clic. Dans ce cas, on peut dire que le document d'origine est *interactif*.
- En revanche, les références internes posent un défi pour la composition du document. En effet, quiconque a l'expérience de la préparation de documents scientifiques et techniques moyennement longs sait qu'il est presque inévitable que la numérotation des pages, des sections, des images, des tableaux, des théorèmes ou d'autres éléments similaires à ce qui est indiqué dans la référence, change au cours de la préparation du document, ce qui rend très difficile sa mise à jour.

Avant l'avènement de l'informatique, les auteurs évitaient les références internes ; et celles qui étaient inévitables, comme la table des matières (qui, si elle est accompagnée du numéro de page de chaque section, est un exemple de référence interne), étaient écrites à la toute fin.

Même les systèmes de composition les plus limités, comme les traitements de texte, permettent l'inclusion d'une certaine forme de références croisées internes, comme les tables des matières. Mais ce n'est rien comparé au mécanisme complet de gestion des références inclus dans ConT_EXt, qui peut également combiner le mécanisme de gestion des références internes visant à maintenir les références à jour, avec l'utilisation d'hyperliens qui n'est évidemment pas exclusive aux références externes.

9.2 Références internes

Deux choses sont nécessaires pour établir une référence interne :

1. Une étiquette ou un identifiant au point cible. Lors de la compilation, ConTeXt, associera des données particulières à cette étiquette. Les données associées dépendent du type d'étiquette ; il peut s'agir du numéro de section, du numéro de note, du numéro d'image, du numéro associé à un élément particulier dans une liste numérotée, du titre de la section, etc.
2. Une commande au point d'origine qui lit les données associées à l'étiquette liée au point cible et les insère au point d'origine. La commande varie en fonction des données de l'étiquette que l'on souhaite insérer au point d'origine.

Lorsque nous pensons à une référence, nous le faisons en termes de « origine → cible », il pourrait donc sembler que les questions relatives à l'origine doivent être expliquées en premier, et ensuite celles relatives à la cible. Cependant, je crois qu'il est plus facile de comprendre la logique des références si l'explication est inversée.

9.2.1 L'étiquette de la cible

Dans ce chapitre, par *étiquette*, j'entends une chaîne de texte qui sera associée au point cible et utilisée en interne pour y faire référence et ainsi récupérer certaines informations concernant le point cible (par exemple le numéro de page, le numéro de section, etc). En fait, les informations associées à chaque étiquette dépendent de la procédure de création de celle-ci. ConTeXt appelle ces étiquettes *références*, mais je pense que ce dernier terme, ayant un sens beaucoup plus large, est moins clair.

L'étiquette associée à la référence cible :

- a besoin que chaque cible potentielle du document soit unique afin qu'elle puisse être identifiée sans aucun doute. Si nous utilisons la même étiquette pour différentes cibles, ConTeXt ne lancera pas d'erreur de compilation (il génèrera des messages), mais toutes les références pointeront vers la première étiquette qu'il trouvera (dans le fichier source) et cela aura pour effet secondaire que certaines de nos références pourront être erronées et, pire encore, que nous ne les remarquerons pas. Il est donc important de s'assurer, lors de la création d'une étiquette, que la nouvelle étiquette que nous attribuons n'a pas déjà été attribuée auparavant.
- Il peut contenir des lettres, des chiffres, des signes de ponctuation, des espaces vides, etc. Lorsqu'il y a des espaces vides, les règles générales de ConTeXt concernant ces types de caractères s'appliquent toujours (voir [section 4.2.1](#)), de sorte que, par exemple, « Mon beau libellé » et « Mon beau libellé » sont considérés comme identiques, même si un nombre différent d'espaces vides est utilisé dans les deux.

Puisqu'il n'y a aucune limitation quant aux caractères pouvant faire partie de l'étiquette et à leur nombre, mon conseil est d'utiliser des noms d'étiquette clairs, qui

nous aideront à comprendre le fichier source lorsque, peut-être, nous le lirons longtemps après qu'il ait été écrit. C'est pourquoi l'exemple que j'ai donné précédemment (« Mon beau libellé ») n'est pas un bon exemple, car il ne nous dit rien sur la cible vers laquelle pointe l'étiquette. Pour cette rubrique, par exemple, l'étiquette « `sec:EtiquettesCibles` » serait meilleure (« `sec` » indiquant qu'il s'agit d'une section).

Pour associer une cible particulière à une étiquette, il existe essentiellement deux procédures :

1. Au moyen d'un argument ou d'une option de commande utilisée pour créer l'élément vers lequel l'étiquette pointera. De ce point de vue, toutes les commandes qui créent une sorte de structure ou d'élément textuel susceptible d'être une cible de référence comprennent une option nommée « `reference` » qui est utilisée pour inclure l'étiquette. Parfois, à la place de l'option, l'étiquette est le contenu de l'argument entier.

Nous trouvons un bon exemple de ce que j'essaie de dire dans les commandes de section qui, comme nous le savons depuis (section ??), permettent plusieurs types de syntaxe. Voici comment définir une étiquette avec les deux syntaxes :

Dans la syntaxe classique, la commande est écrite comme suit :

```
\section[étiquette]{Titre de la section}

\startsection
  [title=Titre de la section,
   reference=étiquette]
\stopsection
```

Dans les deux cas, la commande prévoit l'introduction d'une étiquette qui sera associée à la section (ou chapter, subsection, etc.) en question.

J'ai dit que cette fonctionnalité se retrouve dans *toutes les commandes* qui nous permettent de créer un élément de texte susceptible d'être la cible d'une référence. Il s'agit de tous les éléments de texte qui peuvent être numérotés, y compris, entre autres, les sections, les objets flottants de toutes sortes (tableaux, images et autres), les notes de bas de page ou de fin, les citations, les listes numérotées, les descriptions, les définitions, etc.

Lorsque l'étiquette est saisie directement avec un argument, et non comme une option à laquelle une valeur est attribuée, il est possible avec ConT_EXt d'associer plusieurs étiquettes à une seule cible. Par exemple :

```
\section[étiquette1,étiquette2,étiquette3]{Titre de la section}
```



Je ne vois pas bien quel serait l'avantage d'avoir plusieurs étiquettes différentes pour la même cible et je soupçonne que cela peut être fait non pas parce que cela offre des avantages, mais en raison d'une exigence *interne* de ConT_EXt applicable à certains types d'arguments.

2. Au moyen des commandes `\pagereference`, `\reference`, ou `\textreference` dont la syntaxe est :

```
Texte initial \pagereference[étiquette]
\reference[étiquette]{Texte}
\textreference[étiquette]{Texte}
et texte final
```

Texte initial et texte final

- L'étiquette créée avec `\pagereference` nous permet de récupérer le numéro de page.
- Les étiquettes créées avec `\reference` et `\textreference` nous permettent de récupérer le numéro de page ainsi que le texte qui leur est associé en argument.

Dans les deux cas `\reference` et `\textreference`, le texte lié à l'étiquette n'apparaît pas en tant que tel dans le document final au point où se trouve la commande, mais il peut être récupéré et réapparaître au point d'origine de la référence.

J'ai dit précédemment que chaque étiquette est associée à certaines informations concernant le point cible. La nature de ces informations dépend du type d'étiquette :

- Toutes les étiquettes *rappellent* (dans le sens où elles permettent de les retrouver) le numéro de page de la commande qui les a créées. Pour les étiquettes attachées à des sections qui peuvent avoir plusieurs pages, ce numéro sera celui de la page où commence la section en question.
- Les étiquettes insérées avec la commande qui crée un élément textuel numéroté (section, note, tableau, image, etc.) *rappellent* le numéro associé à cet élément (numéro de section, numéro de note, etc.)
- Si les éléments possèdent un *titre*, comme c'est le cas, par exemple, pour les sections, mais aussi les tableaux s'ils ont été insérés à l'aide de la commande `\placetable`, les étiquettes se *rappelleront* de ces titres.
- Les étiquettes créées avec `\pagereference` *rappellent* le numéro de page.
- Celles créées avec `\reference` ou `\textreference` *rappellent* également le texte qu'elles prennent comme argument.

En fait, je ne suis pas sûr de la différence réelle entre les commandes `\reference` et `\textreference`. Je pense qu'il est possible que la conception des trois commandes qui permettent la création d'étiquettes tente de fonctionner en parallèle avec les trois commandes qui permettent la récupération d'informations à partir des étiquettes (que nous verrons dans un moment) ; mais la vérité est que, selon mes tests, `\reference` et `\textreference` semblent être des commandes redondantes.

9.2.2 Commandes au point de référence d'origine pour récupérer les données du point cible

Les commandes que je vais vous expliquer ensuite récupèrent les informations des étiquettes et, en plus, si notre document est interactif, génèrent un lien hypertexte

vers la cible. Mais ce qui est important dans ces commandes, c'est l'information qui est récupérée de l'étiquette. Si nous voulons seulement générer l'hyperlien, sans récupérer aucune information de l'étiquette, nous devons utiliser la commande `\goto` expliquée dans la [section 9.4.2](#).

A. Commandes de base pour récupérer les informations d'une étiquette

Sachant que chaque étiquette associée à un point cible peut stocker différents éléments d'information, il est logique que ConTeXt comprenne trois commandes différentes pour récupérer ces informations : selon les informations d'un point cible de référence que nous voulons récupérer, nous utilisons l'une ou l'autre de ces commandes :

- La commande `\at` permet de récupérer le numéro de page de l'étiquette.
- Pour les étiquettes qui mémorisent un numéro d'élément (numéro de section, numéro de note, numéro d'article, numéro de table, etc.) en plus du numéro de page, la commande `\in` permet de récupérer ce numéro.
- Enfin, pour les étiquettes qui mémorisent un texte associé à une étiquette (titre de section, titre d'image inséré avec `\placefigure`, etc.), la commande `\about` permet de récupérer ce texte.

Les trois commandes `\at` `\in` `\about` ont les syntaxes suivantes :

```
\setupinteraction[state=start]
\startsubsection
  [title=Titre de la section,
   reference=sec:cettesection]
\stopsubsection

À la \at{page}[sec:cettesection]
se trouve la \in{section}{.}[sec:cettesection]
dont le titre est \about[sec:cettesection]
```

1 Titre de la section

À la **page 0** se trouve la **section 1.** dont le titre est "**Titre de la section**"

- « `sec:cettesection` » est l'étiquette de la cible dont nous voulons récupérer l'information, c'est un point commun entre toutes les commandes.
- le premier texte entre accolades sera ajouté juste avant l'information que l'on souhaite récupérer avec la commande. Entre le texte et les données de l'étiquette que la commande récupère, un espace insécable sera inséré et si la fonction d'interactivité est activée (avec `\setupinteraction[state=start]`) l'information récupérée par la commande génère un lien hypertexte qui nous dirigera au point cible. Le texte inclus entre accolades fera partie de ce lien (il sera cliquable).
- dans le cas de `\in` il est possible d'indiquer un second texte entre accolades, celui-ci sera ajouté après l'information récupérée par la commande, et sera également intégré au lien hypertexte. Cela est utile par exemple pour générer un hyperlien « 1- Section » au lieu de « Section 1 ».

Ainsi, dans l'exemple précédent, nous voyons comment `\in` récupère le numéro de section et `\at` le numéro de page et `\about` le titre.

Notez que ConTeXt a automatiquement créé des hyperliens (voir section ??), et que le texte pris comme argument par `\in` et `\at` fait partie du lien. Mais si nous l'avions écrit autrement, le résultat serait :

```
\setupinteraction[state=start]
\startsubsection
  [title=Titre de la section,
   reference=sec:cettesection]
\stopsubsection

À la page \at{}[sec:cettesection]
se trouve la section \in{}{}[sec:cettesection]
dont le titre est \about[sec:cettesection]
```

1 Titre de la section

À la page 0 se trouve la section 1 dont le titre est “Titre de la section”

Le texte reste le même, mais les mots *section* et *page* qui précèdent la référence ne sont pas inclus dans le lien car ils ne font plus partie de la commande.

Si ConTeXt n'est pas en mesure de trouver l'étiquette vers laquelle les commandes `\at`, `\in` ou `\about` pointent, aucune erreur de compilation n'en résultera mais là où les informations récupérées par ces commandes devraient apparaître dans le document final, nous verrons écrit « ?? ».

Il y a deux raisons pour lesquelles ConTeXt ne peut pas trouver une étiquette :

1. Nous avons fait une erreur en l'écrivant.
2. Nous ne compilons qu'une partie du document, et l'étiquette pointe vers la partie non encore compilée (voir sections 4.5.1 et 4.6).

Dans le premier cas, l'erreur devra être corrigée. C'est pourquoi, lorsque nous aurons fini de compiler le document complet (et que le deuxième cas ne sera plus possible), il est bon de rechercher toutes les apparitions de « ?? » dans le PDF pour vérifier qu'il n'y a pas de références cassées dans le document.

B. Récupération des informations associées à une étiquette avec la commande `\ref`

Chacune des commandes `\at`, `\in` et `\about` récupère certains éléments d'une étiquette. Il existe une autre commande qui nous permet de récupérer un élément de l'étiquette indiquée. Il s'agit de la commande `\ref` dont la syntaxe est :

```
\ref [Élément à Récupérer] [Étiquette]
```

où le premier argument peut être :

- `text` : renvoie le texte associé à une étiquette.
- `title` : renvoie le titre associé à un libellé.
- `number` : renvoie le numéro associé à un libellé. Par exemple, dans les sections, le numéro de section.

- `page` : renvoie le numéro de la page.
- `realpage` : renvoie le numéro de la page d'un point de vue physique (car parfois la numérotation des pages ne commence qu'à partir de l'introduction).
- `default` : renvoie ce que ConTeXt considère comme l'élément *natural* de l'étiquette. En général, cela coïncide avec ce qui est renvoyé par `number`.

En fait, `\ref` peut permettre d'être plus précis que `\at`, `\in` ou `\about`, et donc, par exemple, il fait la différence entre le numéro de page et le numéro de page réel. Le numéro de page peut ne pas coïncider avec le numéro réel si, par exemple, la numérotation des pages du document a commencé à 1500 (parce que ce document est la suite d'un précédent) ou si les pages du préambule étaient numérotées en chiffres romains et que, voyant cela, la numérotation a été recommencée. De même, `\ref` fait la différence entre le *text* et le *title* associés à une référence, ce que `\about`, par exemple, ne permet pas.

Si `\ref` est utilisé pour obtenir des informations à partir d'une étiquette qui n'en possède pas (par exemple, le titre d'une étiquette associée à une note de bas de page), la commande renverra une chaîne vide.

<pre> \setupinteraction[state=start] \startsubsection [title=Titre de la section, reference=sec:cettesection] \stopsubsection {\tt text :} \ref [text] [sec:cettesection] \\ {\tt title :} \ref [title] [sec:cettesection] \\ {\tt number :} \ref [number] [sec:cettesection] \\ {\tt page :} \ref [page] [sec:cettesection] \\ {\tt realpage :} \ref [realpage] [sec:cettesection] </pre>	<pre> 1 Titre de la section text : title : Titre de la section number : 1 page : 0 realpage : 1 </pre>
--	---

C. Détecter où le lien mène

ConTeXt possède également deux commandes qui sont sensibles à *l'adresse du lien*. Avec l'« adresse du lien », mon intention est de déterminer si la cible du lien dans le fichier source se trouve avant ou après l'origine. Par exemple : nous sommes en train de rédiger notre document et nous voulons faire référence à une section qui pourrait se trouver avant ou après celle que nous sommes en train d'écrire dans la table des matières finale. Nous n'avons pas encore décidé. Dans cette situation, il serait utile de disposer d'une commande qui écrit l'un ou l'autre selon que la cible se situe finalement avant ou après l'origine dans le document final. Pour des besoins de ce type, ConTeXt fournit la commande `\somewhere` dont la syntaxe est :

```
\somewhere{texte si avant}{texte si après} [Étiquette].
```


Par exemple :

```
\setupinteraction[state=start]
Dans une \somewhere
{section précédente} {section prochaine}
[sec:cettesection].

\startsubsection
[title=Titre de la section,
reference=sec:cettesection]
\stopsubsection

Dans une \somewhere
{section précédente} {section prochaine}
[sec:cettesection].
```

Dans une **section prochaine**.

1 Titre de la section

Dans une **section précédente**.

Une autre commande capable de détecter si l'étiquette qu'elle pointe vient avant ou après, est `\atpage` dont la syntaxe est :

```
\atpage[label]
```

Cette commande est assez similaire à la précédente, mais au lieu de nous permettre d'écrire nous-mêmes le texte, selon que l'étiquette se trouve avant ou après, `\atpage` insère un texte par défaut pour chacun des deux cas et, si le document est interactif, insère également un lien hypertexte.

Le texte que `\atpage` insère est celui associé à l'option « hencefore » de la commande `\setuplabeltext`, dans le cas où l'étiquette de la cible se trouve *avant* la commande `\atpage` (et celui associé à l'option « hereafter » dans le cas contraire).

Lorsque je suis arrivé à ce point, j'ai été trahi par une décision antérieure : dans ce chapitre, j'ai décidé d'appeler ce que ConTeXt appelle une « référence », une « étiquette ». Cela me semblait plus clair. Mais certains fragments de texte générés par les commandes ConTeXt tels que `\atpage`, sont également appelés « labels » (cette fois dans un autre sens). (Voir [section 10.5.3](#)). J'espère que le lecteur ne sera pas perdu par ma façon de présenter. Je pense que le contexte nous permet de distinguer correctement à laquelle des différentes significations de *étiquette* je fais référence dans chaque cas.

Par conséquent, nous pouvons modifier le texte inséré par `\atpage` de la même manière que nous modifions le texte de toute autre étiquette :

```

\mainlanguage[fr]
\setupinteraction[state=start]
\setuplabeltext[fr]
[hereafter=Comme nous le montrerons plus
tard]
\setuplabeltext[fr]
[hencefore=Comme nous l'avons vu avant]

\atpage[sec:cettesection].
\startsubsection
  [title=Titre de la section,
   reference=sec:cettesection]
\stopsubsection
\atpage[sec:cettesection].

```

Comme nous le montrerons plus tard.

1 Titre de la section

Comme nous l'avons vu avant.

9.2.3 Génération automatique de préfixes pour éviter les étiquettes en double

Dans un document volumineux, il n'est pas toujours facile d'éviter la duplication des étiquettes. Il est donc conseillé de mettre un peu d'ordre dans la façon dont nous choisissons les étiquettes à utiliser. Une bonne pratique consiste à utiliser des préfixes pour les étiquettes qui varient en fonction du type d'étiquette. Par exemple, « sec: » pour les sections, « fig: » pour les figures, « tbl: » pour les tableaux, etc.

Dans cette optique, ConT_EXt inclut une collection d'outils qui permettent :

- de générer automatiquement des étiquettes pour tous les éléments autorisés.
- de faire en sorte que chaque étiquette générée manuellement prenne un préfixe, soit celui que nous avons prédéterminé nous-mêmes, soit celui généré automatiquement par ConT_EXt.

L'explication détaillée de ce mécanisme est longue et, bien qu'il s'agisse sans aucun doute d'outils utiles, je ne pense pas qu'ils soient indispensables. Par conséquent, comme ils ne peuvent être expliqués en quelques mots, je préfère ne pas les expliquer et renvoyer à ce qui est dit à leur sujet dans le manuel de référence ConT_EXt ou dans le [wiki](#) sur cette question.

9.3 Documents électroniques interactifs

Seuls les documents électroniques peuvent être interactifs, mais tous les documents *électroniques* ne le sont pas. Un document électronique est un document qui est stocké dans un fichier informatique et qui peut être ouvert et lu directement à l'écran. En revanche, un document électronique équipé de fonctionnalités qui permettent à l'utilisateur d'*interagir* avec lui est interactif, c'est-à-dire qu'on peut faire plus que le lire. Il y a interactivité, par exemple, lorsque le document comporte des boutons permettant d'effectuer une action, ou des liens permettant de passer à un autre point du document, ou à un document externe ; ou lorsque le document comporte des zones où l'utilisateur peut écrire, ou des vidéos ou des clips audio qui peuvent être lus, etc.

Tous les documents générés par ConT_EXt sont électroniques (puisque ConT_EXt génère un PDF qui est par définition un document électronique), mais ils ne sont pas toujours interactifs. Pour leur conférer une interactivité, il est nécessaire de l'indiquer expressément, comme le montre la section suivante.

Il faut cependant savoir que, même si ConT_EXt génère un PDF interactif, pour apprécier cette interactivité, nous avons besoin d'un lecteur de PDF capable de le faire, car tous les lecteurs de PDF ne nous permettent pas d'utiliser des hyperliens, des boutons et autres éléments similaires propres aux documents interactifs.

9.3.1 Permettre l'interactivité dans les documents

ConT_EXt n'utilise pas de fonctions interactives par défaut, sauf indication expresse, ce qui est normalement fait dans le préambule du document. La commande qui active cet utilitaire est :

```
\setupinteraction[state=start]
```

Normalement, cette commande ne devrait être utilisée qu'une seule fois et dans le préambule du document lorsque nous voulons générer un document interactif. Mais en fait, nous pouvons l'utiliser aussi souvent que nous le souhaitons en modifiant l'état d'interactivité du document. La commande « `state=start` » active l'interactivité, tandis que « `state=stop` » la désactive, de sorte que nous pouvons désactiver l'interactivité dans certains chapitres ou *parties* de notre document comme nous le souhaitons.

Je ne vois aucune raison pour laquelle nous voudrions avoir des parties non interactives dans des documents qui sont interactifs. Mais ce qui est important dans la philosophie de ConT_EXt c'est que quelque chose soit techniquement possible, même s'il est peu probable que nous l'utilisions, et elle offre donc une procédure pour le faire. C'est cette philosophie qui donne à ConT_EXt tant de possibilités, et empêche une simple introduction comme celle-ci d'être *bref*.

Une fois l'interaction établie :

- Certaines commandes ConT_EXt incluent naturellement des hyperliens telles que :

- Les commandes de création de tables des matières : en cliquant sur une entrée de la table des matières, on accède à la page où commence la section en question.
- Les commandes de références internes que nous avons vues dans la première partie de ce chapitre : cliquer dessus permet de sauter automatiquement à la cible de la référence.
- Les notes de bas de page et notes de fin de texte où un clic sur l’ancrage de la note dans le corps du texte principal nous amènera à la page où la note elle-même est écrite, et un clic sur la marque de la note dans le texte de la note nous amènera au point du texte principal où l’appel a été fait.
- idem pour les index, etc.
- D’autres commandes sont activées car spécialement conçues pour les documents interactifs, comme les présentations. Celles-ci utilisent de nombreux outils associés à l’interactivité tels que des boutons, des menus, des superpositions d’images, des sons ou des vidéos intégrés, etc. L’explication de tout cela serait trop longue et de plus, les présentations sont un type de document assez particulier. C’est pourquoi, dans les lignes qui suivent, je décrirai une seule fonctionnalité associée à l’interactivité : les hyperliens.

9.3.2 Configuration de base pour les éléments interactifs

`\setupinteraction`, en plus d’activer ou de désactiver l’interaction, nous permet de configurer certains éléments qui y sont liés ; principalement, mais pas seulement, la couleur et le style des liens. Cela se fait par le biais des options de commande suivantes :

- `color` : contrôle la couleur *par défaut* des liens.
- `contrastcolor` : détermine la couleur des liens lorsque leur cible se trouve sur la même page que l’origine. Je recommande que cette option soit toujours définie sur le même contenu que la précédente.
- `style` : contrôle le style du lien.
- `title`, `subtitle`, `author`, `date`, `keyword` : Les valeurs attribuées à ces options seront converties en métadonnées du PDF généré par ConTeXt.
- `click` : Cette option contrôle si le lien doit être mis en évidence lorsqu’il est cliqué.

9.4 Hyperliens vers des documents externes

Je distinguerai les commandes qui ne créent pas le lien mais aident à composer l'URL du lien, et les commandes qui créent l'hyperlien à proprement parler. Examinons-les séparément :

9.4.1 Commandes de définition d'hyperlien (sans les introduire dans le document)

Les URL ont tendance à être très longues, et comprennent des caractères de tous types, même des caractères réservés en ConTeXt et qui ne peuvent pas être utilisés directement. De plus, lorsque l'URL doit être affichée dans le document, il est très difficile de composer le paragraphe, car l'URL peut dépasser la longueur d'une ligne et ne comporte jamais d'espaces vides pouvant être utilisés pour insérer un saut de ligne. De plus, dans une URL, il n'est pas raisonnable de césumer les mots pour insérer des sauts de ligne, car le lecteur pourrait difficilement savoir si la césure fait ou non partie de l'URL.

C'est pourquoi ConTeXt fournit deux utilitaires pour *la composition* des URL. Le premier est principalement destiné aux URL qui seront utilisées en interne, mais qui ne seront pas réellement affichées dans le document. Le second est destiné aux URL qui doivent être écrits dans le texte du document. Examinons-les séparément :

`\useURL`

Cette commande nous permet d'écrire une URL dans le préambule du document, en l'associant à un nom, de sorte que lorsque nous voulons l'utiliser ensuite dans notre document, nous pouvons l'invoquer par le nom qui lui est associé. Elle est particulièrement utile pour les URL qui seront utilisées plusieurs fois dans le document.

Cette commande permet deux utilisations :

1. `\useURL[NomAssocié] [URL]`
2. `\useURL[NomAssocié] [URL] [] [Texte du lien]`

- Dans la première version, l'URL est simplement associée au nom par lequel elle sera invoquée dans notre document. Mais alors, pour utiliser l'URL, nous devons indiquer d'une manière ou d'une autre, en l'invoquant, quel texte cliquable sera affiché dans le document.

Dans la deuxième version, le dernier argument inclut le texte cliquable. Le troisième argument existe dans le cas où nous voulons diviser une URL en deux parties, de sorte que la première partie contienne l'adresse d'accès et la deuxième partie le nom du document ou de la page spécifique que nous voulons ouvrir. Par exemple, avec l'adresse du document qui explique ce qu'est ConTeXt:

<pre> \setupinteraction[state=start] \useURL [WhatIsCTXa] [http://www.pragma-ade.com/general/manuals/what-is-context.pdf] [What is \ConTeXt?] \useURL [WhatIsCTXb] [http://www.pragma-ade.com/general/manuals/what-is-context.pdf] [What is \ConTeXt?] \url[WhatIsCTXa] \ \url[WhatIsCTXb] \from[WhatIsCTXa] </pre>	<pre> http://www.pragma-ade.com/general/manuals/what-is-context.pdf http://www.pragma-ade.com/general/manuals/what-is-context.pdf What is ConTeXt? </pre>
--	---

Dans les deux cas, nous aurons associé cette adresse au mot « WhatIsCTX ». Si, à un moment quelconque du texte, nous voulons reproduire une URL que nous avons associée à un nom à l'aide de la commande `\useURL`, nous pouvons utiliser la commande `\url[NomAssocié]` qui insère l'URL associée à ce nom dans le document. Mais cette commande, bien qu'elle écrive l'URL, ne crée aucun lien. La commande `\from` insère le texte associé au lien (voir section 9.4.2).

Le format d'affichage des URL écrites à l'aide de `\url` n'est pas celui établi de manière générale au moyen de `\setupinteraction`, mais celui établi spécifiquement pour cette commande au moyen de `\setupurl`, qui nous permet de définir le style (option `style`) et la couleur (option `color`).

`\hyphenatedurl`

Cette commande est destinée aux URL qui seront écrits dans le texte de notre document, et ConTeXt doit inclure des sauts de ligne dans l'URL, si nécessaire, afin de composer correctement le paragraphe. Son format est le suivant :

```
\hyphenatedurl{AdresseURL}
```

Malgré le nom de la commande `\hyphenatedurl`, elle ne met pas de trait d'union dans le nom de l'URL. Ce qu'elle fait, c'est considérer que certains caractères courants dans les URL sont de bons points pour insérer un saut de ligne avant ou après eux. Nous pouvons ajouter les caractères que nous voulons à la liste des caractères pour lesquels un saut de ligne est autorisé. Nous disposons de trois commandes pour cela :

```

\sethyphenatedurlnormal{caractères}
\sethyphenatedurlbefore{caractères}
\sethyphenatedurlafter{caractères}

```

Ces commandes ajoutent, respectivement, les caractères qu'elles prennent comme arguments à la liste des caractères qui supportent les sauts de ligne avant et après, uniquement avant, et uniquement après.

`\hyphenatedurl` peut être utilisée lorsqu'il faut écrire une URL qui apparaîtra telle quelle dans le document final. Elle peut même être utilisée comme dernier argument de `\useURL` dans la version de cette commande où le dernier argument récupère le texte cliquable à afficher dans le document final.

Dans l'argument `\hyphenatedurl`, tous les caractères réservés peuvent être utilisés, sauf trois qui doivent être remplacés par des commandes :

- % doit être remplacé par `\letterpercent`
- # doit être remplacé par `\letterhash`
- \ doit être remplacé par `\letterescape` ou `\letterbackslash`.

Chaque fois que `\hyphenatedurl` insère un saut de ligne, il exécute la commande `\hyphenatedurlseparator`, qui, par défaut, ne fait rien. Mais si nous la redéfinissons, un caractère représentatif est inséré dans l'URL de manière similaire à ce qui se passe avec les mots normaux, où un trait d'union est inséré pour indiquer que le mot continue sur la ligne suivante. Par exemple :

```
\setupinteraction[state=start]
\useURL [WhatIsCTXa]
[http://www.pragma-ade.com/general/manuals]
[what-is-context.pdf]
[http://www.pragma-ade.com/general/manuals/what-is-context.pdf]

\useURL [WhatIsCTXb]
[http://www.pragma-ade.com/general/manuals]
[what-is-context.pdf]
[\hyphenatedurl[http://www.pragma-ade.com/general/manuals/what-is-context.pdf]]

\from[WhatIsCTXa]\\
\from[WhatIsCTXb]\\
\def\hyphenatedurlseparator{\curvearrowright}
\from[WhatIsCTXb]
```

<http://www.pragma-ade.com/general/manuals/what-is-context.pdf>
<http://www.pragma-ade.com/general/manuals/what-is-context.pdf>
<http://www.pragma-ade.com/general/manuals/what-is-context.pdf>

9.4.2 Commandes d'insertion d'hyperlien dans le document

Pour établir des liens vers des URL prédéfinis à l'aide de `\useURL`, nous pouvons utiliser la commande `\from`, qui se limite à établir le lien, mais n'écrit aucun texte cliquable. Le texte par défaut de `\useURL` sera utilisé comme texte du lien. Sa syntaxe est la suivante :

```
\from[Nom]
```

où *Nom* est le nom précédemment associé à une URL à l'aide de `\useURL`.

```
\setupinteraction[state=start]
\useURL [WhatIsCTX]
[http://www.pragma-ade.com/general/manuals]
[what-is-context.pdf]
[What is \ConTeXt]

\from[WhatIsCTX]
```

What is ConTeXt

Pour créer des liens et les associer à un texte cliquable qui n'a pas été préalablement défini, nous disposons de la commande `\goto` qui est utilisée à la fois pour générer des liens internes et externes. Sa syntaxe est la suivante :

```
\goto{Texte utilisé en lien}[Cible]
```

où *Texte utilisé en lien* est le texte à afficher dans le document final et sur lequel un clic de souris conduira le lecteur à la cible, et *Cible* peut être :

- Une étiquette de notre document. Dans ce cas, `\goto` générera le saut de la même manière que, par exemple, les commandes `\in` ou `\at` déjà examinées. Mais contrairement à ces commandes, aucune information associée à l'étiquette ne sera récupérée.
- L'URL elle-même. Dans ce cas, il faut indiquer expressément qu'il s'agit d'une URL en écrivant la commande comme suit :

```
\goto{Texte utilisé en lien}[url(URL)]
```

où URL, à son tour, peut être le nom précédemment associé à une URL au moyen de `\useURL`, ou l'URL elle-même, auquel cas, lors de l'écriture de l'URL, nous devons nous assurer que les caractères réservés de ConTeXt sont écrits correctement dans ConTeXt. L'écriture de l'URL selon les règles de ConTeXt n'affectera pas la fonctionnalité du lien.

9.5 Création de signets dans le PDF final

Les fichiers PDF peuvent avoir une liste de signets internes qui permet au lecteur de voir le contenu du document dans une fenêtre spéciale du programme de visualisation PDF, et de se déplacer dans le document en cliquant simplement sur chacune des sections et sous-sections.

Par défaut, ConT_EXt ne fournit pas au PDF de sortie une liste de signets, mais pour qu'il le fasse, il suffit d'inclure la commande `\placebookmarks`, dont la syntaxe est :

```
\placebookmarks[ListeDeSections] [SectionOuverteparDéfaut]
```

où *ListeDeSections* est une liste des niveaux de section, séparées par des virgules, qui doivent apparaître dans la table des matières. *SectionOuverteparDéfaut*, optionnel, indique le niveau de section que le visualisateur de PDF mettre en évidence par défaut (les autres niveaux étant *repliés*). Gardez à l'esprit les observations suivantes concernant cette commande :

- Les types de section définis par l'utilisateur (avec `\definehead`) ne sont pas toujours situés au bon endroit dans la liste des signets. Il est préférable de les exclure.
- Si le titre d'une section comprend une note de fin ou de bas de page, le texte de la note de bas de page est considéré comme faisant partie du signet.
- En guise d'argument, au lieu d'une liste de sections, on peut simplement indiquer le mot symbolique « all » qui, comme son nom l'indique, inclura toutes les sections ; cependant, d'après mes tests, ce mot, en plus de ce qui est certainement des sections, inclut des textes placés là avec certaines commandes de non-sectionnement, de sorte que la liste résultante est quelque peu imprévisible.

Tous les programmes de lecture de PDF ne nous permettent pas d'afficher les signets ; et beaucoup de ceux qui le font n'ont pas cette fonction activée par défaut. Par conséquent, pour vérifier le résultat de cette fonction, nous devons nous assurer que notre programme de lecture de PDF prend en charge cette fonction et qu'elle est activée. Je crois me souvenir qu'Acrobat, par exemple, n'affiche pas les signets par défaut, bien qu'il existe un bouton dans sa barre d'outils pour les afficher.

9.6 Pièces jointes

Pour finir avec les fonctionnalités d’interactivité de ConT_EXt, il permet d’introduire des pièces attachées dans vos documents avec la commande `\attachment`.

```
\attachment  
[Nom de la pièce jointe]  
[file={fichier.txt},  
author={auteur de la pièce jointe}]
```

Le fichier attaché doit être dans le même répertoire que le fichier `.tex` dans lequel se trouve la commande `\attachment` ou bien être dans l’un des répertoires indiqués avec la commande `\usepath` (voir [section 4.6.4](#)).

III

Composition des éléments locaux

Chapitre 10

Caractères, mots, texte et espace horizontal

Table of Contents: 10.1 Obtenir des caractères qui ne sont pas normalement accessibles à partir du clavier; 10.1.1 Diacritics and special letters; 10.1.2 Traditional ligatures; 10.1.3 Greek letters; 10.1.4 Various symbols; 10.1.5 Defining characters ; 10.1.6 Use of predefined symbol sets; 10.2 Special character formats; 10.2.1 Upper case, lower case and fake small caps; 10.2.2 Superscript or subscript text; 10.2.3 Verbatim text; 10.3 Character and word spacing; 10.3.1 Automatically setting horizontal space; 10.3.2 Altering the space between characters within a word; 10.3.3 Commands for adding horizontal space between words; 10.4 Compound words; 10.5 The language of the text; 10.5.1 Setting and changing the language; 10.5.2 Configuring the language; 10.5.3 Labels associated with particular languages; 10.5.4 Some language-related commands; A Date-related commands; B The `\translate` command; C The `\quote` and `\quotation` commands;

L'élément de base de tous les documents textuels est le caractère : les caractères sont regroupés en mots, qui à leur tour forment des lignes qui constituent les paragraphes qui composent les pages.

Le présent chapitre, qui commence par « *caractère* », explique certains des utilitaires de ConTeXt relatifs aux caractères, aux mots et au texte.

10.1 Obtenir des caractères qui ne sont pas normalement accessibles à partir du clavier

Dans un fichier texte codé en UTF-8 (voir section ??), nous pouvons utiliser n'importe quel caractère ou symbole, aussi bien des langues vivantes que de nombreuses langues dites mortes. Mais, comme les possibilités d'un clavier sont limitées, la plupart des caractères et symboles autorisés en UTF-8 ne peuvent normalement pas être obtenus directement du clavier. C'est notamment le cas de nombreux diacritiques, c'est-à-dire des signes placés au-dessus (ou au-dessous) de certaines lettres, leur conférant une valeur particulière ; mais aussi de nombreux autres caractères comme les symboles mathématiques, les ligatures traditionnelles, etc. Nous pouvons obtenir beaucoup de ces caractères avec ConTeXt en utilisant des commandes.

10.1.1 Diacritics and special letters

¹⁶ Of the commands found in [table 10.1](#) the tilde does not work with the letter «e», and I don't know why.

¹⁷ Remember that in this document we are representing blank spaces, when it is important that we see them, with the «`_`».

Almost all Western languages have diacritics (with the important exception of English for the most part) and in general, keyboards can generate the diacritics corresponding to regional languages. Thus, a Spanish keyboard can generate all the diacritics needed for Spanish (basically accents and diaeresis) as well as some diacritics used in others languages such as Catalan (grave accents and cedillas) or French (cedillas, grave and circumflex accents); but not, for example, some that are used in Portuguese, such as the tilde on some vowels in words like «navegação».

TeX was designed in the United States where keyboards generally do not enable us to get diacritics; so Donald Knuth gave it a set of commands that enable us to obtain almost all the known diacritics (at least in languages using the Latin alphabet). If we use a Spanish keyboard, it does not make much sense to use these commands to obtain the diacritics that can be obtained directly from the keyboard. It is still important to know that these commands exist, and what they are, since Spanish (or Italian, or French...) keyboards do not let us generate all possible diacritics.

Name	Character	Abbreviation	Command
Acute accent	ú	<code>\'u</code>	<code>\uacute</code>
Grave accent	ù	<code>\`u</code>	<code>\ugrave</code>
Circumflex accent	û	<code>\^u</code>	<code>\ucircumflex</code>
Dieresis or umlaut	ü	<code>\"u</code>	<code>\udiaeresis</code> , <code>\uumlaut</code>
Tilde	ũ	<code>\~u</code>	<code>\utilde</code>
Macron	ū	<code>\=u</code>	<code>\umacron</code>
Breve	ǔ	<code>\u u</code>	<code>\ubreve</code>

Tableau 10.1 Accents and other diacritics

In [table 10.1](#) we find the commands and abbreviations that allow us to obtain these diacritics. In all cases it is unimportant whether we use the command or the abbreviation. In the table, I have used the letter «u» as an example, but these commands work with any vowel (most of them¹⁶) and also with some consonants and some semivowels.

- As most of the abbreviated commands are *control symbols* (see [section 3.2](#)), the letter on which the diacritic is to fall can be written immediately after the command, or separated from it. So, for example: to get the Portuguese «ã» we can write the `\=a` or `\=_a` characters.¹⁷ But in the case of the breve (`\u`), when dealing with a *control word* the blank space is obligatory.
- In the case of the long version of the command, the letter on which the diacritic falls will be the first letter of the command name. So, for example `\emacron` will place a macron above a lower case «e» (ē), `\Emacron` will do the same above an upper case «E» (Ē), while `\Amacron` will do the same above an upper case «A» (Ā).

While the commands in [table 10.1](#) work with vowels and some consonants, there are other commands to generate some diacritics and special letters which only work on one or several letters. They are shown in [table 10.2](#).

Name	Character	Abbreviation	Command
Scandinavian O	ø, Ø	\o, \O	
Scandinavian A	å, Å	\aa, \AA, {\r a}, {\r A}	\aring, \Aring
Polish L	ł, Ł	\l, \L	
German Eszett	ß	\ss, \SS	
«i» and «j» without a point	ı, Ĳ	\i, \j	
Hungarian Umlaut	ű, Ű	\H u, \H U	
Cedilla	ç, Ç	\c c, \c C	\ccedilla, \Ccedilla

Tableau 10.2 More diacritics and special letters

I would like to point out that some of the commands in the above table generate the characters from other characters, while other commands only work if the font we are using has expressly provided for the character in question. So where German Eszett (ß) is concerned, the table shows two commands but only one character, because the font I am using here for this text only provides for the upper case version of German Eszett (something quite common).

That's probably why I can't get the Scandinavian A in upper case either although «{\r A}» and \Aring work correctly.

The Hungarian umlaut also works with the letter «o», and the cedilla with the letters «k», «l», «n», «r», «s» and «t», in lower or upper case, respectively. The commands to be used are \kcedilla, \lcedilla, \ncedilla ... respectively.

10.1.2 Traditional ligatures

A ligature is formed by the union of two or more graphemes that are usually written separately. This «fusion» between two characters often started out as a kind of shorthand in handwritten texts, until finally they achieved a certain typographic independence. Some of them were even included among the characters that are usually defined in a typographic font, such as the ampersand, «&», which began as a contraction of the Latin copula (conjunction) «et», or the German Eszett (ß), which, as its name indicates, began as a combination of an «s» and «z». In some font designs, even today, we can trace the origins of these two characters; or maybe I see them because I know they're there. In particular, with the Pagella font for «&» and with Bookman for «ß».

As an exercise I suggest (after reading Chapter ??, where it explains how to do it) try representing these characters with these fonts at a size large enough (for example, 30 pt) to be able to work out their components.

Other traditional ligatures which did not become so popular, but are still used occasionally today, are the Latin endings «oe» and «ae» which were occasionally written as «œ» or «æ» to indicate that they formed a diphthong in Latin. These ligatures can be achieved in ConTeXt with the commands found in [table 10.3](#)

Ligature	Abbreviation	Command
æ, Æ	\ae, \AE	\aeligature, \AEligature
œ, Œ	\oe, \OE	\oeligature, \OEligature

Tableau 10.3 Traditional ligatures

¹⁸ In L^AT_EX, by contrast, we can use the \DH command implemented by the «fontenc» package.

A ligature that used to be traditional in Spanish (Castilian) and that is not usually found in fonts today, is «Ð»: a contraction involving «D» and «E». As far as I know there is no command in ConT_EXt that lets us use this,¹⁸ but we can create one, as explained in [section 10.1.5](#).

Along with the previous ligatures, which I have called *traditional* because they come from handwriting, after the invention of the printing press certain printed text ligatures developed which I will call «typographical ligatures» considered by ConT_EXt to be font utilities and which are managed automatically by the program, although we can influence how these font utilities are handled (including ligatures) with `\definefontfeature` (not explained in this introduction).

10.1.3 Greek letters

It is common to use Greek characters in mathematical and physics formulas. This is why ConT_EXt included the possibility of generating all of the Greek alphabet, upper and lower case. Here the command is built on the English name for the Greek letter in question. If the first character is written in lower case we will have the lower case Greek letter and if it is written in capital letters we will get the Greek letter in upper case. For example, the command `\mu` will generate the lower case version of this letter (μ) while `\Mu` will generate the upper case version (M). In [table 10.4](#) we can see which command generates each of the letters in the Greek alphabet, lower case and upper case.

English name	Character (lc/uc)	Commands (lc/uc)
Alpha	α , A	<code>\alpha</code> , <code>\Alpha</code>
Beta	β , B	<code>\beta</code> , <code>\Beta</code>
Gamma	γ , Γ	<code>\gamma</code> , <code>\Gamma</code>
Delta	δ , Δ	<code>\delta</code> , <code>\Delta</code>
Epsilon	ϵ , ε , E	<code>\epsilon</code> , <code>\varepsilon</code> , <code>\Epsilon</code>
Zeta	ζ , Z	<code>\zeta</code> , <code>\Zeta</code>
Eta	η , H	<code>\eta</code> , <code>\Eta</code>
Theta	θ , ϑ , Θ	<code>\theta</code> , <code>\vartheta</code> , <code>\Theta</code>
Iota	ι , I	<code>\iota</code> , <code>\Iota</code>
Kappa	κ , κ , K	<code>\kappa</code> , <code>\kappa</code> , <code>\Kappa</code>
Lambda	λ , Λ	<code>\lambda</code> , <code>\Lambda</code>
Mu	μ , M	<code>\mu</code> , <code>\Mu</code>
Nu	ν , N	<code>\nu</code> , <code>\Nu</code>
Xi	ξ , Ξ	<code>\xi</code> , <code>\Xi</code>
Omicron	o , O	<code>\omicron</code> , <code>\Omicron</code>
Pi	π , ϖ , Π	<code>\pi</code> , <code>\varpi</code> , <code>\Pi</code>
Rho	ρ , ϱ , P	<code>\rho</code> , <code>\varrho</code> , <code>\Rho</code>
Sigma	σ , ς , Σ	<code>\sigma</code> , <code>\varsigma</code> , <code>\Sigma</code>
Tau	τ , T	<code>\tau</code> , <code>\Tau</code>
Ypsilon	υ , Y	<code>\upsilon</code> , <code>\Upsilon</code>
Phi	ϕ , φ , Φ	<code>\phi</code> , <code>\varphi</code> , <code>\Phi</code>
Chi	χ , X	<code>\chi</code> , <code>\Chi</code>
Psi	ψ , Ψ	<code>\psi</code> , <code>\Psi</code>
Omega	ω , Ω	<code>\omega</code> , <code>\Omega</code>

Tableau 10.4 Greek alphabet

Note how for lower case versions of some characters (epsilon, kappa, theta, pi, rho, sigma and phi) there are two possible variants.

10.1.4 Various symbols

Together with the characters we have just seen, \TeX (and therefore \ConTeXt as well) offers commands for generating any number of symbols. There are many such commands. I have provided an extended although incomplete list in appendix ??.

10.1.5 Defining characters

If we need to use any characters not accessible from our keyboard, we can always find a web page with these characters and copy them into our source file. If we are using UTF-8 encoding (as recommended) this will almost always work. But also in the \ConTeXt wiki there is a page with heaps of symbols that can be simply copied and pasted into our document. To get them, click [on this link](#).

However, if we need to use one of the characters in question more than once, then copy-paste is not the most efficient way to do so. It would be preferable to define the character so that it is associated with a command that will generate it each time. To do this we use \definecharacter whose syntax is:

```
 $\text{\definecharacter}$  Name Character
```

where

- **Name** is the name associated with the new character. It should not be the name of an existing command, as this would overwrite that command.
- **Character** is the character generated each time we run \Name . There are three ways we can indicate this character:
 - By simply writing it or pasting it into our source file (if we have copied it from another electronic document or web page).
 - By indicating the number associated with that character in the font we are currently using. In order to see the characters included in the font, and the numbers associated with them, we can use the $\text{\showfont[Font name]}$ command.
 - Building the new character with one of the composite character building commands that we will see immediately following.

As an example of the first usage, let's return for the moment to the sections dealing with ligatures (10.1.2). There I spoke about a traditional ligature in Spanish that we can't usually find in fonts today: «D». We could associate this character, for example, with the \decontract command so that the character will be generated whenever we write \decontract . We do this with:

```
 $\text{\definecharacter}$  decontract D
```

To build a new character that is not in our font, and cannot be obtained from the keyboard, as is the case of the example I have just given, first we must find some text where that character is found, copy it and be able to paste it into our definition. In the actual example I have just given, I originally copied the «D» from Wikipedia.

ConT_EXt also includes some commands that allow us to create composite characters and that can be used in combination with `\definecharacter`. By composite characters I mean characters that also have diacritics. The commands are as follows:

```
\buildmathaccent Accent Character
\buildtextaccent Accent Character
\buildtextbottomcomma Character
\buildtextbottomdot Character
\buildtextcedilla Character
\buildtextgrave Character
\buildtextmacron Character
\buildtextogonek Character
```

For example: as we already know, by default ConT_EXt only has commands for writing certain letters with a cedilla (c, k, l, n, r, s y t) that are usually incorporated into fonts. If we wanted to use a «b» we could use the `\buildtextcedilla` command as follows:

```
\definecharacter bcedilla {\buildtextcedilla b}
```

This command will create the new `\bcedilla` command that will generate a «b» with a cedilla: «ḅ». These commands literally «build» the new character that will be generated even though our font doesn't have it. What these commands do is to superimpose one character over another then give a name to that superimposition.

In my tests I was unable to make `\buildmathaccent` or `\buildtextogonek` work. So I will no longer mention them from here on.

`\buildtextaccent` takes two characters as arguments and superimposes one on the other, raising one of them slightly. Although it is called «buildtextaccent», it is not essential that any of the characters taken as arguments is an accent; but the overlap will give better results if it is, because in this case, by superimposing the accent on the character the accent is less likely to overwrite the character. On the other hand, the overlapping of two characters that have the same baseline under normal conditions is affected by the fact that the command slightly raises one of the characters above the other. This is why we cannot use this command, for example, to get the contraction «Đ» mentioned above, because if we write

```
\definecharacter decontract {\buildtextaccent D E}
```

in our source file, the slight elevation above the «D» baseline that this command produces means that the («Đ») effect it produces is not very good. But if the height of the characters allows it we could create a combination. For example,

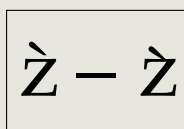
```
\definecharacter unusual {\buildtextaccent \_ "}
```

would define the «_» character that would be associated with the `\unusual` command.

The rest of the build commands takes a single argument – the character that the diacritic generated by each command will be added to. Below I will show an example of each of them, built on the letter «z»:

- `\buildtextbottomcomma` adds a comma beneath the character it takes as an argument (« $\underset{\sim}{z}$ »).
- `\buildtextbottomdot` adds a point beneath the character it takes as an argument (« $\underset{\cdot}{z}$ »).
- `\buildtextcedilla` adds a cedilla beneath the character it takes as an argument (« $\underset{\text{~}}{z}$ »).
- `\buildtextgrave` adds a grave accent above the character it takes as an argument (« \grave{z} »).
- `\buildtextmacron` adds a small bar beneath the character it takes as an argument (« $\underset{\bar}{z}$ »).

At first sight, `\buildtextgrave` seems redundant given that we have `\buildtextaccent`; However, if you check the grave accent generated with the first of these two commands, it looks a little better. The following example shows the result of both commands, at a sufficient font size to appreciate the difference:



10.1.6 Use of predefined symbol sets

« ConT_EXt Standalone » includes, along with ConT_EXt itself, a number of predefined symbol sets we can use in our documents. These sets are called « `cc` », « `cow` », « `fontawesome` », « `jmn` », « `mvs` » and « `nav` ». Each of these sets also includes some subsets:

- `cc` includes « `cc` ».
- `cow` includes « `cownormal` » and « `cowcontour` ».
- `fontawesome` includes « `fontawesome` ».
- `jmn` includes « `navigation 1` », « `navigation 2` », « `navigation 3` » and « `navigation 4` ».
- `mvs` includes « `astronomic` », « `zodiac` », « `europe` », « `martinvogel 1` », « `martinvogel 2` » and « `martinvogel 3` ».
- `nav` includes « `navigation 1` », « `navigation 2` » and « `navigation 3` ».

The wiki also mentions a set called `wasy` that includes « `wasy general` », « `wasy music` », « `wasy astronomy` », « `wasy astrology` », « `wasy geometry` », « `wasy physics` » and « `wasy apl` ». But I couldn't find them in my distribution, and my tests to attempt to get at them failed.

To see the specific symbols contained in each of these sets, the following syntax is used:

```
\usesymbols[Set]
```

```
\showsymbolset [Subset]
```

For example: if we want to see the symbols included in « mvs/zodiac », then in the source file we need to write:

```
\usesymbols [mvs]  
\showsymbolset [zodiac]
```

and we will get the following result:

Aquarius
Aries
Cancer
Capricorn
Gemini
Leo
Libra
Pisces
Sagittarius
Scorpio
Taurus
Virgo

Note that the name of each symbol is indicated as well as the symbol. The `\symbol` command allows us to use any of the symbols. Its syntax is:

```
\symbol [Subset] [SymbolName]
```

where subset is one of the subsets associated with any of the sets we have previously loaded with `\usesymbols`. For example, if we wanted to use the astrological symbol associated with Aquarius (found in mvs/zodiac) we would need to write

```
\usesymbols [mvs]  
\symbol [zodiac] [Aquarius]
```

which will give us the « » symbol, and this, for all intents and purposes, will be treated as a « character » and is therefore affected by the font size that is active when printed. We can also use `\definecharacter` to associate the symbol in question with a command. For example

```
\definecharacter Aries {\symbol [zodiac] [Aries]}
```

will create a new command called `\Aries` that will generate the character « ».

We could also use these symbols, for example, in an itemize environment. For example:

```
\usesymbols [mvs]  
\definesymbol [1] [{\symbol [martinvogel 2] [PointingHand]}]  
\definesymbol [2] [{\symbol [martinvogel 2] [CheckedBox]}]  
\startitemize [packed]  
\item item \item item  
\startitemize [packed]  
\item item \item item
```

```
\stopitemize
\item item
\stopitemize
```

will produce

- item
- item
 - item
 - item
- item

10.2 Special character formats

Strictly speaking, it is *format* commands that affect the font used, its size, style or variant. These commands are explained in Chapter ???. However, seen more *broadly*, we can also consider the commands that somehow change the characters they take as an argument (thus altering their appearance) to be format commands. We will look at some of these commands in this section. Others, such as underlined or lined text with lines above or below the text (e.g. where we want to provide space to answer a question) will be seen in section ???.

10.2.1 Upper case, lower case and fake small caps

Letters themselves can be upper case or lower case. For ConT_EXt, upper case and lower case letters are different characters, so in principle it will typeset the letters just as it finds them written. However, there is a group of commands which allow us to ensure that the text they take as an argument is always written in upper or lower case:

- `\word{text}`: converts the text taken as an argument into lower case.
- `\Word{text}`: converts the first letter of the text taken as an argument into upper case.
- `\Words{text}`: converts the first letter of each of the words taken as an argument into upper case; the rest are in lower case.
- `\WORD{text}` or `\WORDS{text}`: writes the text taken as an argument in upper case.

Very similar to these commands are `\cap` and `\Cap`: they also capitalise the text they take as an argument, but then apply a scaling factor to it equal to that applied by the «x» suffix in font change commands (see [section 6.4.2](#)) so that, in most fonts, the caps will be the same height as lower case letters, thus giving us a kind of *fake small caps* effect. Compared to genuine small caps (see [section 6.5.2](#)) these have the following advantages:

1. `\cap` and `\Cap` will work with any font, by contrast with genuine small caps that only work with fonts and styles that expressly include them.
2. True small caps, on the other hand, are a variant of the font which, as such, is incompatible with any other variant such as bold, italic, or slanted. However, `\cap` and `\Cap` are fully compatible with any font variant.

The difference between `\cap` and `\Cap` is that while the former applies the scaling factor to all the letters of the words that make up its argument, `\Cap` does not apply any scaling to the first letter of each word, thus achieving an effect similar to what we get if we use real capitals in a text in small caps. If the text taken as an argument in «caps» consists of several words, the size of the capital letter in the first letter of each word will be maintained.

Thus, in the following example

<code>The UN, whose \Cap{president} has his office at \cap{uN} headquarters...</code>	The UN, whose @president has his office at UN headquarters...
---	--

we need to note, first of all, the difference in size between the first time we write « UN » (in upper case) and the second time (in small caps, « UN »). In the example, I wrote `\cap{uN}` the second time so we can see that it does not matter if we write the argument that `\cap` takes in upper or lower case: the command converts all letters into upper case and then applies a scaling factor; by contrast with `\Cap` that does not scale the first letter.

These commands can also be *nested*, in which case the scaling factor would be applied once more, resulting in a further reduction, as in the following example where the word « capital » in the first line is scaled yet again:

<code>\cap{People who have amassed their \cap{capital} at the expense of others are more often than not \bf decapitated} in revolutionary times}.</code>	PEOPLE WHO HAVE AMASSED THEIR CAPITAL AT THE EX- PENSE OF OTHERS ARE MORE OFTEN THAN NOT DECA- PITATED IN REVOLUTIONARY TIMES.
--	--

The `\nocap` command applied to a text to which `\cap` is applied, cancels out the `\cap` effect in the text that is its argument. For example:

<code>\cap{When I was One I had just begun, when I was Two I was \nocap{nearly} new (A.A. Milne)}.</code>	WHEN I WAS ONE I HAD JUST BEGUN, WHEN I WAS TWO I WAS nearly NEW (A.A. MILNE).
---	---

We can configure how `\cap` works with `\setupcapitals` and we can also define different versions of the command, each with its own name and specific configuration. This we can do with `\definecapitals`.

Both commands work in a similar way:

```
\definecapitals[Name] [Configuration]
\setupcapitals[Name] [Configuration]
```

The « Name » parameter in `\setupcapitals` is optional. If it is not used, the configuration will affect the `\cap` command itself. If it is used, we need to give the name we previously assigned in `\definecapitals` to some actual configuration.

In either of the two commands the configuration allows for three options: « title », « sc » and « style » the first and second allowing « yes » and « no » as values. With « title » we indicate whether the capitalisation will also affect titles (which it does by default) and with « sc » we indicate whether the command should be genuine small caps (« yes »), or fake small caps (« no »). By default it uses fake small caps which has the advantage that the command works even if you are using a font that

has not implemented small caps. The third value « `style` » allows us to indicate a style command to be applied to the text affected by the `\cap` command.

10.2.2 Superscript or subscript text

We already know (see [section 3.1](#)) that in maths mode, the reserved characters « `_` » and « `^` » will convert the character or group that immediately follows into a superscript or subscript. To achieve this effect outside of maths mode, ConTeXt includes the following commands:

- `\high{Text}`: writes the text it takes as an argument as a superscript.
- `\low{Text}`: writes the text it takes as an argument as a subscript.
- `\lohi{Subscript}{Superscript}`: writes both arguments, one above the other: on the bottom the first argument, and on top the second, which brings about a curious effect:

<code>\lohi{below}{above}</code>		above below
----------------------------------	--	----------------

10.2.3 Verbatim text

The Latin expression *verbatim* (from *verbum* = *word* + the suffix *atim*), which could be translated as « literally » or « word for word », is used in text processing programs like ConTeXt to refer to fragments of text that should not be processed at all, but should be dumped, as written, into the final file. ConTeXt uses the command `\type` for this, intended for short texts that do not occupy more than one line and the typing environment intended for texts of more than one line. These commands are widely used in computer books to show code fragments, and ConTeXt formats these texts in monospaced letters like a typewriter or a computer terminal would. In both cases the text is sent to the final document without *processing*, which means that they can use reserved characters or special characters that will be transcribed *as is* in the final file. Likewise, if the argument of `\type`, or the content of `\starttyping` includes a command, this will be *written* in the final document, but not executed.

The `\type` command has, besides, the following peculiarity: its argument *can* be contained within curly brackets (as is normal in ConTeXt), but any other character can be used to delimit (surround) the argument.

When ConTeXt reads the `\type` command it assumes that the character which is not a blank space immediately following the name of the command will act as a delimiter of its argument; so it considers that the contents of the argument begin with the next character, and end with the character before the next appearance of the *delimiter*.

Some examples will help us to understand this better:

```
\type 1Tweedledum and Tweedledee1
\type |Tweedledum and Tweedledee|
\type zTweedledum and Tweedledeez
\type (Tweedledum and Tweedledee(
```

Note that in the first example, the first character after the command name is a «1», in the second a «|» and in the third a «z»; so: in each of these cases ConTeXt will consider that the

argument of `\type` is everything between that character and the next appearance of the same character. The same is true for the last example, which is also very instructive, because in principle we could assume that if the opening delimiter of the argument is a «(», the closing one should be a «)», but it is not, because «(» and «)» are different characters and `\type`, as I said, searches for a closing character delimiter which is the same as the character used at the start of the argument.

There are only two cases where `\type` allows the opening and closing delimiters to be different characters:

- If the opening delimiter is the «{» character, it thinks the closing delimiter will be «}».
- If the opening delimiter is «<<», it thinks that the closing delimiter will be «>>». This case is also unique in that two consecutive characters are being used as delimiters.

However: the fact that `\type` allows any delimiter does not mean that we should use « weird » delimiters. From the point of view of the *readability* and *comprehensibility* of the file source, it is best to delimit the argument of `\type` with curly brackets where possible, as is normal with ConT_EXt; and when this is not possible, because there are curly brackets in the `\type` argument, use a symbol: preferably one that is not a ConT_EXt reserved character. For example: `\type *This is a closing curly bracket: '}'*`.

Both `\type` and `\starttyping` can be configured with `\setuptype` and `\setuptyping`. We can also create a customised version of these with `\definetype` and `\definetyping`. Regarding the actual configuration options for these commands, I refer to « `setup-en.pdf` » (in the directory `tex/texmf-context/doc/context/documents/general/qrcs`).

Two very similar commands to `\type` are:

- `\typ`: works similarly to `\type`, but does not disable hyphenation.
- `\tex`: a command intended for writing texts about T_EX or ConT_EXt: it adds a backspace before the text it takes as an argument. Otherwise, this command differs from `\type` in that it processes some of the reserved characters it finds in the text it takes as an argument. In particular, curly brackets inside `\tex` will be treated in the same way they are usually treated in ConT_EXt.

¹⁹ It is very typical of the philosophy of ConTeXt to include a command to do something that the ConTeXt documentation itself advises against doing. Although typographical perfection is sought, the aim is also to give the author absolute control over the appearance of his or her document: whether it is better or worse is, in short, his or her responsibility.

10.3 Character and word spacing

10.3.1 Automatically setting horizontal space

The space between different characters and words (called *horizontal space* in TeX) is normally set automatically by ConTeXt:

- The space between the characters that make up a word is defined by the font itself, which, except in fixed-width fonts, usually uses a greater or lesser amount of white space depending on the characters to be separated, and so, for example, the space between an «A» and a «V» («AV») is usually less than the space between an «A» and an «X» («AX»). However, apart from these possible variations that depend on the combination of letters concerned and predefined by the font, the space between the characters that make up a word is, in general, a fixed and invariable measure.
- By contrast, the space between words on the same line can be more elastic.
 - In the case of words in a line whose width must be the same as that of the rest of the lines in the paragraph, the variation of the spacing between words is one of the mechanisms that ConTeXt uses to obtain lines of equal width, as explained in more detail in section ???. In these cases, ConTeXt will establish exactly the same horizontal space between all the words in the line (except for the rules below), while ensuring that the space between words in the different lines of the paragraph is as similar as possible.
 - However, in addition to the need to stretch or shrink the spacing between words in order to justify the lines, depending on the active language, ConTeXt takes certain typographical rules into consideration whereby in certain places the typographical tradition associated with that language adds some extra white space, as is the case, for example, in some parts of the English typographical tradition, which adds extra white space after a full stop.

These extra white spaces work for English and possibly for some other languages (though it is also true that in many instances, publishers in English nowadays choose not to have extra space after a full stop) but not for Spanish where the typographical tradition is different. So we can temporarily enable this function with `\setupspacing[broad]` and disable it with `\setupspacing[packed]`. We could also change the default configuration for Spanish (and for that matter for any other language including English), as explained in [section 10.5.2](#).

10.3.2 Altering the space between characters within a word

Altering the default space for the characters that make up a word is considered very bad practice from a typographical point of view, except in titles and headings. However, ConTeXt provides a command to alter this space between the characters in a word:¹⁹ `\stretched`, whose syntax is as follows:

`\stretched[Configuration]{Text}`

where *Configuration* allows any of the following options:

- **factor**: an integer or decimal number representative of the spacing to be obtained. It should not be too high a number. A factor of 0.05 is already visible to the naked eye.
- **width**: indicates the total width that the text submitted to the command must have, in such a way that the command itself will calculate the necessary spacing to distribute the characters in that space.

According to my tests, when the width established with the `width` option is less than that required to represent the text with a *factor* equal to 0.25, the `width` option and this factor are ignored. I guess that's because `\stretched` allows us only to *increase* the space between the characters in a word, not reduce it. But I don't understand why the width required to represent the text with a factor of 0.25 is used as a minimum measure for the `width` option, and not the *natural width* of the text (with a factor of 0).

- **style:** style command or commands to apply to the text taken as an argument.
- **color:** the colour in which the text taken as an argument will be written.

So in the following example we can see graphically how the command would work when applied to the same sentence, but with different widths:

```
\stretched[width=4cm]{\bf test text}  
\stretched[width=6cm]{\bf test text}  
\stretched[width=8cm]{\bf test text}  
\stretched[width=9cm]{\bf test text}
```

[illegible]

In this example it can be seen that the distribution of the horizontal space between the different characters is not uniform. The «x» and «t» in « text » and the «e» and «b» in « test », always appear much closer together than the other characters. I haven't been able to find out why this happens.

Applied without arguments, the command will use the full width of the line. On the other hand, within the text that is the argument to this command, the command `\hspace` is redefined and instead of a line break, it inserts horizontal space. For example:

`\stretched{test\\text}` test text

We can customise the default configuration of the command with `\setupstretched`.

There is no `\definestretched` command that would allow us to set customised configurations associated with a command name, however, in the official list of commands (see [section 3.6](#)) it says that `\setupstretched` comes from `\setupcharacterkerning` and there is a `\definecharacterkerning` command. In my tests, however, I have not managed to set any customised configuration for `\stretched` by means of the latter, although I must admit that I have not spent much time trying to do so either.



10.3.3 Commands for adding horizontal space between words

We already know that to increase the space between words it is of no use to add two or more consecutive blank spaces, since ConTeXt absorbs all consecutive blank spaces, as explained in [section 4.2.1](#). If we wish to increase the space between words, we need to go to one of the commands that allows us to do this:

- `\,` inserts a very small blank space (called a thin space) in the document. It is used, for example, to separate thousands in a set of numbers (e.g. 1,000,000), or to separate a single inverted comma from double inverted commas. For example: «`1\,473\,451`» will produce «1 473 451».
- `\space` or «`_`» (a backslash followed by a blank space which, since it is an invisible character, I have represented as «`_`») introduces an additional blank space.
- `\enskip`, `\quad` and `\qquad` insert a blank space in the document of half an *em*, 1 *em* or 2 *ems* respectively. Remember that the *em* is a measure dependent on the size of the font and is equivalent to the width of an «m», which normally coincides with the size in points of the font. So, using a 12 point font, `\enskip` gives us a space of 6 points, `\quad` gives us 12 points and `\qquad` gives us 24 points.

Along with these commands which give us blank space in precise measurements, the `\hskip` and `\hfill` commands introduce horizontal space of varying dimensions:

`\hskip` allows us to indicate exactly how much blank space we want to add. Thus:

<code>This is \hskip 1cm 1 centimetre\\</code>		This is	1 centimetre
<code>This is \hskip 2cm 2 centimetres\\</code>		This is	2 centimetres
<code>This is \hskip 2.5cm 2.5 centimetres\\</code>		This is	2.5 centimetres

The space indicated may be negative, which will cause one text to be superimposed over another. Thus:

<code>This is farce rather than</code>		This is farce rather than
<code>\hskip -1cm comedy</code>		comedy

`\hfill`, for its part, introduces as much white space as necessary to occupy the entire line, allowing us to create interesting effects such as right-aligned text, centred text or text on both sides of the line as shown in the following example:

<code>\hfill On the right\\</code>		On the right
<code>On both\hfill sides</code>		On both sides

10.4 Compound words

By « compound words » in this section I mean words that are formally understood to be one word, rather than words that are simply conjoined. It is not always an easy distinction to understand: « rainbow » is clearly made up of two words (« rain + bow ») but no English speaker would think of the combined terms in any other way than as a single word. On the other hand, we have words that are sometimes combined with the help of a hyphen or backslash. The two words have distinct meanings and uses but are conjoined (and may in some cases become a single word, but not yet!). So, for example, we can find words like « French–Canadian » or « (inter)communication » (though we may well also find « intercommunication » and discover that the speaking public has finally accepted the two words to be a single word. That is how language evolves).

Compound words present ConTeXt with some problems mainly connected with their potential hyphenation at the end of a line. If the joining element is a hyphen, then from a typographical perspective there is no hyphenation problem at the end of a line at that point, but we would need to avoid a second hyphenation in the second part of the word since that would leave us with two consecutive hyphens which could cause comprehension difficulties.

The « || » command is available to tell ConTeXt that two words make up a compound word. This command, exceptionally, does not begin with a backslash, and allows two different usages:

- We can use two consecutive vertical bars (pipes) and write, for example, « Spanish| |Argentine ».
- The two vertical bars can have the joining /separating item between two words enclosed between them, as in, for example, « joining|/|separating ».

In both cases, ConTeXt will know that it is dealing with a compound word, and will apply the appropriate hyphenation rules for this type of word. The difference between using the two consecutive vertical bars (pipes), or framing the word separator with them, is that in the first case, ConTeXt will use the separator that is predefined as `\setuphyphenmark`, or in other words the hyphen, which is the default (« -- »). So if we write « picture| |frame », ConTeXt will generate « Picture–frame ».

With `\setuphyphenmark` we can change the default separator (in the case where we need two pipes). The values allowed for this command are « --, ---, -, , , (,), =, / ». Bear in mind, however, that the « = » value becomes an em dash (the same as « --- »).

The normal use of « || » is with hyphens, since this is what is normally used between composite words. But occasionally the separator could be a parenthesis, if, for example, we want « (inter)space », or it could be a forward slash, as in « input/output ». In these cases, if we want the normal hyphenation rules for composite words to apply, we could write « (inter|) |space » or « input|/|output ». As I said earlier, « |=| » is considered to be an abbreviation of « |---| » and inserts an em dash as a separator (—).

²⁰ Table 10.5 has a summary of the list obtained with the following commands:

```
\usemodule[languages-system]
\loadinstalledlanguages
\showinstalledlanguages
```

Should you be reading this document long after it was written (2020) it is possible that ConT_EXt will have incorporated additional languages, so it would be a good idea to use these commands to show an updated list of languages

10.5 The language of the text

Characters form words which normally belong to some language. It is important for ConT_EXt to know the language we are writing in, because a number of important things depend on this. Mainly:

- Word hyphenation.
- The output format of certain words.
- Certain typesetting matters associated with the typesetting tradition of the language in question.

10.5.1 Setting and changing the language

ConT_EXt assumes that the language will be English. Two procedures can change this:

- By using the `\mainlanguage` command, used in the preamble to change the main language of the document.
- By using the `\language` command, aimed at changing the active language at any point in the document.

Both commands expect an argument consisting of any language identifier (or code). To identify the language, we use either the two-letter international language code set out in ISO 639-1, which is the same as that used, for example, on the web, or the English name of the language in question, or sometimes some abbreviation of the name in English.

In table 10.5 we find a complete list of languages supported by ConT_EXt, along with the ISO code for each of the languages in question as well as, where appropriate, the code for certain language variants expressly provided for.²⁰

So, for example, to set Spanish (Castilian) as the main language of the document we could use any of the three that follow:

```
\mainlanguage[es]
\mainlanguage[spanish]
\mainlanguage[sp]
```

To enable a particular language *inside* the document, we can use either the `\language[Language code]` command, or a specific command to activate that language. So, for example, `\en` activates English, `\fr` activates French, `\es` Spanish, or `\ca` Catalan. Once an actual language has been activated, it remains so until we expressly switch to another language, or the group in which the language was activated is then closed. Languages work, therefore, just like font change commands. Note, however, that the language set by the `\language` command or by one of its abbreviations (`\en`, `\fr`, `\de`, etc.) does not affect the language in which labels are printed (see section 10.5.3).

Although it may be laborious to mark the language of all the words and expressions we use in our document that do not belong to the main language of the document, it is important to

Language	ISO code	Language (variants)
Afrikaans	af, afrikaans	
Arabic	ar, arabic	ar-ae, ar-bh, ar-dz, ar-eg, ar-in, ar-ir, ar-jo, ar-kw, ar-lb, ar-ly, ar-ma, ar-om, ar-qa, ar-sa, ar-sd, ar-sy, ar-tn, ar-ye
Catalan	ca, catalan	
Czech	cs, cz, czech	
Croatian	hr, croatian	
Danish	da, danish	
Dutch	nl, nld, dutch	
English	en, eng, english	en-gb, uk, ukenglish, en-us, usenglish
Estonian	et, estonian	
Finnish	fi, finnish	
French	fr, fra, french	
German	de, deu, german	de-at, de-ch, de-de
Greek	gr, greek	
Greek (ancient)	agr, ancientgreek	
Hebrew	he, hebrew	
Hungarian	hu, hungarian	
Italian	it, italian	
Japanese	ja, japanese	
Korean	kr, korean	
Latin	la, latin	
Lithuanian	lt, lithuanian	
Malayalam	ml, malayalam	
Norwegian	nb, bokmal, no, norwegian	nn, nynorsk
Persian	pe, fa, persian	
Polish	pl, polish	
Portuguese	pt, portuguese	pt-br
Romanian	ro, romanian	
Russian	ru, russian	
Slovak	sk, slovak	
Slovenian	sl, slovene, slovenian	
Spanish	es, sp, spanish	es-es, es-la
Swedish	sv, swedish	
Thai	th, thai	
Turkish	tr, turkish	tk, turkmen
Ukrainian	ua, ukrainian	
Vietnamese	vi, vietnamese	

Tableau 10.5 Language support in ConT_EXt

do so if we want to obtain a properly typeset final document, especially in professional work. We should not mark all the text, but only the part that does not belong to the main language. Sometimes it is possible to automate the marking of the language by using a macro. For example, for this document in which ConT_EXt commands are continuously being quoted, the original language of which is English, I have designed a macro which, in addition to writing the command in the appropriate format and colour, marks it as an English word. In my professional work, where I need to quote a lot of French and Italian bibliography, I have incorporated a

field in my bibliographic database to pick up the language of the work, so that I can automate the language indication in the quotations and lists of bibliographical references.

If we are using two languages that use different alphabets in the same document (for example, English and Greek, or English and Russian), there is a trick that will prevent us from having to mark the language of expressions built with the alternative alphabet: modify the main language setting (see next section) so that it also loads the default hyphenation patterns for the language that uses a different alphabet. For example, if we want to use English and ancient Greek, the following command would save us from having to mark language of the texts in Greek:

```
\setuplanguage[en][patterns={en, agr}]
```

This only works because English and Greek use a different alphabet, so there can be no conflict in the hyphenation patterns of the two languages, therefore we can load them both simultaneously. But in two languages that use the same alphabet, loading the hyphenation patterns simultaneously will necessarily lead to inappropriate hyphenation.

10.5.2 Configuring the language

ConT_EXt associates the functioning of certain utilities with the specific language active at any given time. The default associations can be changed with `\setuplanguage` whose syntax is:

```
\setuplanguage[Language][Configuration]
```

where *Language* is the language code for the language we want to configure, and *Configuration* contains the specific configuration that we want to set (or change) for that language. Specifically, up to 32 different configuration options are allowed, but I will only deal with those that seem suitable for an introductory text such as this:

- **date:** allows us to configure the default date format. See further ahead on [page 255](#).
- **lefthyphenmin, righthyphenmin:** the minimum number of characters that must be to the left or to the right for hyphenation of a word to be supported. For example `\setuplanguage[en][lefthyphenmin=4]` will not hyphenate any word if there are fewer than 4 characters to the left of the eventual hyphen.
- **spacing:** the possible values for this option are « broad » or « packed ». In the first case (broad), the rules for spacing words in English will be applied, which means that after a full stop and when another character follows, a certain amount of extra blank space will be added. On the other hand, « spacing=packed » will prevent these rules from applying. For English, broad is the default.
- **leftquote, rightquote:** indicate the characters (or commands), respectively, that `\quote` will use to the left and right of the text that is its argument (for this command, see [page 256](#)).
- **leftquotation, rightquotation:** indicate the characters (or commands), respectively that `\quotation` will use to the left and right of the text that is its argument (for this command, see [page 256](#)).

10.5.3 Labels associated with particular languages

Many of ConT_EXt's commands automatically generate certain texts (or *labels*), as, for example, the `\placetable` command that writes the label « Table xx » under the table that is inserted, or `\placefigure` which inserts the label « Figure xx ».

These *labels* are sensitive to the language set with `\mainlanguage` (but not if set with `\language`) and we can change them with

```
\setuplabeltext [Language] [Key=Label]
```

where *Key* is the term by which ConT_EXt knows the label and *Label* is the text we want ConT_EXt to generate. So, for example,

```
\setuplabeltext [es] [figure=Imagen~]
```

would see that when the main language is Spanish, images inserted with `\placefigure` are not called « Figure x » but « Imagen x ». Note that after the text on the label itself, a blank space must be left to ensure that the label is not attached to the next character. In the example I have used the reserved character « ~ »; I could also have written « [figure=Imagen{ }] » enclosing the blank space between curly brackets to ensure that ConT_EXt will not get rid of it.

What labels can we redefine with `\setuplabeltext`? The ConT_EXt documentation is not as complete as one might hope on this point. The 2013 reference manual (which is the one that explains most about this command) mentions « chapter », « table », « figure », « appendix »... and adds « other comparable text elements ». We can assume that the names will be the English names of the element in question.



One of the advantages of *free libre software* is that the source files are available to the user; so we can look into them. I have done so, and *snooping* through the source files of ConT_EXt, I have discovered the file « lang-txt.lua », available in `tex/texmf-context/tex/context/base/mkiv` which I think is the one that contains the predefined labels and their different translations; so that if at any time ConT_EXt generates a redefined text that we want to change, to see the name of the label that text is associated we can open the file in question and find that we want to change. This way we can see which label name is associated with it.

If we want to insert the text associated with a certain label somewhere in the document, we can do so with the `\labeltext` command. So, for example, if I want to refer to a table, to ensure that I name it in the same way that ConT_EXt calls it in the `\placetable` command, I can write: « Just as shown in the `\labeltext{table}` on the next page... » This text, in a document where `\mainlanguage` is English, will produce: « Just as shown in the Tableau on the next page. »

Some of the labels redefinable with `\setuplabeltext`, are empty by default; like, for example, « chapter » or « section ». This is because by default ConT_EXt does not add labels to sectioning commands. If we want to change this default operation, we need only to redefine these labels in the preamble of our document and so, for example, `\setuplabeltext[chapter=Chapter~]` will see that chapters are preceded by the word « Chapter ».

Finally, it is important to point out that although in general, in ConT_EXt, the commands that allow several comma-separated options as an argument, the last option

can end with a comma and nothing bad happens. In `\setuplabeltext` that would generate an error when compiling.

10.5.4 Some language-related commands

A. Date-related commands

ConTeXt has three date-related commands that produce their output in the active language at the time they are run. These are:

- `\currentdate`: run without arguments in a document in which the main language is English, it returns the system date in the format « Day Month Year ». For example: « 11 September 2020 ». But we can also tell it to use a different format (as would happen in the US and some other parts of the English-speaking world that follow their system of putting the month before the day, hence the infamous date, 9/11), or include the name of the day of the week (`weekday`), or include only some elements of the date (`day`, `month`, `year`)

To indicate a different date format, « `dd` » or « `day` » represent the days, « `mm` » the months (in number format), « `month` » the months in alphabetical format in lower case, and « `MONTH` » in upper case. Regarding the year, « `yy` » will write only the last digits, while « `year` » or « `y` » will write all four. If we want some separating element between the date components, we must write it expressly. For example

```
\currentdate[weekday, dd, month]
```

when run on 9 September 2020 will write « Wednesday 9 September ».

- `\date`: this command, run without any argument, produces exactly the same output as `\currentdate`, meaning, the actual date in standard format. However, a specific date can be given as an argument. Two arguments are given for this: with the first argument we can indicate the day (« `d` »), month (« `m` ») and year (« `y` ») corresponding to the date we want to represent, while with the second argument (optional) we can indicate the format of the date to be represented. For example, if we want to know what day of the week John Lennon and Paul McCartney met, an event which, according to Wikipedia, took place on 6 July 1957, we could write

```
\date[d=6, m=7, y=1957][weekday]
```

and so we would find out that such an historical event happened on a Saturday.

- `\month` takes a number as an argument, and returns the name of the month corresponding to that number.

B. The `\translate` command

The `translate` command supports a series of phrases associated with a specific language, so that one or another will be inserted in the final document depending on

the language active at any given time. In the following example, the `translate` command is used to associate four phrases with Spanish and English, which are saved in a memory buffer (regarding the buffer environment, see section ??):

```
\startbuffer
\starttabulate[|*{4}{lw(.25\textwidth)}|]
  \NC \translate[es=Su carta de fecha, en=Your letter dated]
  \NC \translate[es=Su referencia, en=Your reference]
  \NC \translate[es=Nuestra referencia, en=Our reference]
  \NC \translate[es=Fecha, en=Date] \NC\NR
\stoptabulate
\stopbuffer
```

so that if we insert the *buffer* at a point in the document where Spanish is activated, the Spanish phrases will be played, but if the point in the document where the buffer is inserted has English activated, the English phrases will be inserted. Thus:

```
\language[es]
\getbuffer
```

will generate

Su carta de fecha Su referencia Nuestra referencia Fecha

while

```
\language[en]
\getbuffer
```

will generate

Your letter dated Your reference Our reference Date

C. The `\quote` and `\quotation` commands

One of the most common typographical errors in text documents occurs when quote marks (single or double) are opened but not expressly closed. To avoid this happening, ConT_EXt provides the `\quote` and `\quotation` commands that will quote the text that is their argument; `\quote` will use single quotation marks and `\quotation` will use double quotation marks.

These commands are language sensitive in that they use the default character or command set for the language in question to open and close quotes (see [section 10.5.2](#)); and so, for example, if we want to use Spanish as the default style for double quotation marks – the guillemets or chevrons (angle brackets) typical of Spanish, Italian, French, we would write:

```
\setuplanguage[es][leftquotation=«, rightquotation=»].
```

These commands do not, however, manage nested quotes; although we can create the utility that does this, taking advantage of the fact that `\quote` and `\quotation` are actual applications of what ConT_EXt calls *delimitedtext*, and that it is possible to define further applications with `\definedelimitedtext`. Thus the following example:

```
\definedelimitedtext
[CommasLevelA]
[left=«, right=»]

\definedelimitedtext
[CommasLevelB]
[left=", right="]

\definedelimitedtext
[CommasLevelC]
[left=`, right=']
```

will create three commands that will allow up to three different levels of quoting. The first level with side quotes, the second with double quotes and the third with single quotes.

Of course, if we are using English as our main language, then the default single and double quotation marks (curly, not straight, as you find in this document!) will be automatically used.

I Appendices

Annexe A

Index des commandes

Les commandes abordées dans cette introduction sont listées dans cet index. Certaines sont simplement mentionnées, presque en passant, auquel cas la page qui apparaît dans l'index indique où elles sont mentionnées. Mais d'autres commandes font l'objet d'une explication plus détaillée. Dans ce cas, seul l'endroit où commence l'explication détaillée est listé dans l'index, bien que la commande puisse être citée à d'autres endroits de l'introduction également.

Ne sont pas inclus dans cet index :

- Les `\stopQuelqueChose` qui ferment une construction précédemment ouverte avec `\startQuelqueChose`, à moins que le texte ne dise quelque chose de spécial à propos de la commande `\stop`, ou qu'il soit traité à un endroit différent de celui où se trouve la commande `\start` correspondante.
- Les commandes visant à générer des symboles, toutes trouvées dans Appendice ??.
- Dans le cas des commandes visant à générer un diacritique ou une lettre, et qui ont une version majuscule et une version minuscule, pour générer respectivement la majuscule ou la minuscule, seule la version minuscule est incluse.

a	<code>\agrade</code> 235
<code>\aa</code> 236	<code>\alpha</code> 237
<code>\aacute</code> 235	<code>\amacron</code> 235
<code>\about</code> 219	<code>\aring</code> 236
<code>\abreve</code> 235	<code>\at</code> 219
<code>\acircumflex</code> 235	<code>\atilde</code> 235
<code>\adaptlayout</code> 116	<code>\atleftmargin</code> 129
<code>\adaptpapersize</code> 109	<code>\atpage</code> 222
<code>\adiaeresis</code> 235	<code>\atrighmargin</code> 129
<code>\ae</code> 236	
<code>\aeligature</code> 236	

b

\backslash 55
 \begingroup 76, 77
 \beta 237
 \bf 142
 \bfa 143
 \bfb 143
 \bfc 143
 \bfd 143
 \bfx 143
 \bfxx 143
 \bgroup 75, 76
 \bi 142
 \bia 143
 \bib 143
 \bic 143
 \bid 143
 \bigbodyfont 145
 \bix 143
 \bixx 143
 \blank 89
 \bold 142
 \bolditalic 142
 \boldslanted 142
 \bs 142
 \bsa 143
 \bsb 143
 \bsc 143
 \bsd 143
 \bsx 143
 \bsxx 143
 \buildmathaccent 239
 \buildtextaccent 239
 \buildtextbottomcomma 239
 \buildtextbottomdot 239
 \buildtextcedilla 239
 \buildtextgrave 239
 \buildtextmacron 239
 \buildtexttognek 239

c

\Cap 243
 \c 236
 \ca 251
 \calligraphic 142
 \cap 243
 \ccedilla 236
 \cg 142

\chapter 164
 \chi 237
 \color 150, 151
 \colored 151
 \completecontent 190
 \completelistofchemicals 206
 \completelistoffigures 206
 \completelistofgraphics 206
 \completelistofintermezzi 206
 \completelist 205
 \completindex 210
 \completlistoftables 206
 \currentdate 255

d

\date 255
 \de 251
 \define 71
 \definealternativestyle 147
 \definebodyfontenvironment 138, 144
 \definebodyfontswitch 146
 \definecapitals 244
 \definecharacter 238
 \definecharacterkerning 248
 \definecolor 154
 \definecombinedlist 206
 \defineconversion 177
 \defineconversionset 120
 \definedelimitedtext 256
 \ 157
 \definefontfeature 237
 \definefontstyle 146
 \definefontsynonym 158
 \definehead 184
 \definehighlight 150
 \definelayou 117
 \definelist 204
 \definemargindata 132
 \definepapersize 108, 109
 \defineregister 211
 \defineresetset 172
 \definesectionblock 186
 \definestartstop 73
 \definestretched 248
 \definestructureconversionset 175
 \definestructureseparatorset 176
 \definetext 126

`\definetype` 246
`\definetyping` 246
`\delta` 237
`\dontleavehmode` 142

e

`\eacute` 235
`\ebreve` 235
`\ecircumflex` 235
`\ediaeresis` 235
`\egrave` 235
`\egroup` 75, 76
`\em` 148
`\emacron` 235
`\emdash` 91
`\en` 251
`\enableregime` 86
`\endash` 91
`\endgroup` 76, 77
`\enskip` 249
`\environnement` 98
`\epsilon` 237
`\es` 251
`\eta` 237
`\etilde` 235

f

`\fr` 251
`\from` 228

g

`\gamma` 237
`\getmarking` 125
`\goto` 229

h

`\H` 236
`\handwritten` 142
`\hfill` 249
`\high` 245
`\hskip` 249
`\hw` 142
`\hyphen` 91
`\hyphenatedurl` 227
`\hyphenatedurlseparator` 228

i

`\i` 236

`\iacute` 235
`\ibreve` 235
`\icircumflex` 235
`\idiaeresis` 235
`\igrave` 235
`\imacron` 235
`\in` 219
`\index` 208
`\ininner` 129
`\ininneredge` 129
`\ininnermargin` 129
`\inleft` 129
`\inleftedge` 129
`\inleftmargin` 129
`\inmargin` 129
`\inother` 129
`\inouter` 129
`\inouteredge` 129
`\inoutermargin` 129
`\input` 95
`\inright` 129
`\inrightedge` 129
`\inrightmargin` 129
`\iota` 237
`\it` 142
`\ita` 143
`\italic` 142
`\italicbold` 142
`\itb` 143
`\itc` 143
`\itd` 143
`\itilde` 235
`\itx` 143
`\itxx` 143

j

`\j` 236

k

`\kappa` 237
`\kcedilla` 236

l

`\l` 236
`\labeltext` 254
`\lambda` 237
`\language` 251
`\lastpagenumber` 121

`\lastrealpagenumber` 121
`\lastuserpagenumber` 121
`\lcedilla` 236
`\letterbackslash` 228
`\letterescape` 228
`\letterhash` 228
`\letterhat` 55
`\letterpercent` 228
`\lettertilde` 55
`\loadinstalledlanguages` 253
`\lohi` 245
`\low` 245

m

`\mainlanguage` 251
`\margintext` 129
`\mediaeval` 142
`\minus` 91
`\mono` 142
`\month` 255
`\mu` 237

n

`\namedstructurevariable` 167
`\ncedilla` 236
`\nolist` 167, 170
`\nomarking` 167, 170
`\normal` 142
`\nu` 237

o

`\o` 236
`\oacute` 235
`\obreve` 235
`\ocircumflex` 235
`\odiaeresis` 235
`\oe` 236
`\oeligature` 236
`\ograve` 235
`\omacron` 235
`\omega` 237
`\omicron` 237
`\os` 142
`\otilde` 235

p

`\page` 122
`\pagenumber` 121, 125

`\pagereference` 218
`\parskip` 79
`\part` 164
`\phi` 237
`\pi` 237
`\placebookmarks` 230
`\placecontent` 190
`\placeindex` 210
`\placelist` 205
`\placelistofchemicals` 206
`\placelistoffigures` 206
`\placelistofintermezzi` 206
`\placelistofgraphics` 206
`\placelistoftables` 206
`\product` 100
`\project` 101
`\projet` 102
`\psi` 237

q

`\qqquad` 249
`\quad` 249
`\quotation` 256
`\quote` 256

r

`\ReadFile` 96
`\r` 236
`\rcedilla` 236
`\readfile` 96
`\realpagenumber` 121
`\ref` 220
`\reference` 218
`\regular` 142

reserved characters

`\{` 55
`\}` 55
`\$` 55
`\backslash` 55
`\letterhat` 55
`\lettertilde` 55
`\#` 55
`\%` 55
`\&` 55
`_` 55
`\|` 55
`\rho` 237
`\rm` 142

`\rma` 143
`\rmb` 143
`\rmc` 143
`\rmd` 143
`\rmx` 143
`\rmxx` 143
`\roman` 142

s

`\sans` 142
`\sansserif` 142
`\sc` 142
`\scedilla` 236
`\section` 164
`\seeindex` 210
`\serif` 142
`\sethyphenatedurlafter` 227
`\sethyphenatedurlbefore` 227
`\sethyphenatedurlnormal` 227
`\setuparranging` 107, 115
`\setupbackgrounds` 150
`\setupbodyfont` 137, 141
`\setupbottomtexts` 129
`\setupcapitals` 244
`\setupcharacterkerning` 248
`\setupcolors` 150
`\setupcombinedlist` 191
`\setupexternalfigures` 103
`\setupfooter` 126
`\` 126
`\setupfootertexts` 124, 127
`\setuphead` 168
`\setupheader` 126
`\` 126
`\setupheadertexts` 124, 127
`\setupheadnumber` 171
`\setupheads` 168
`\setupheadtext` 190
`\setuphyphenmark` 250
`\setupinteraction` 224
`\setuplabeltext` 254
`\setuplanguage` 253
`\setuplayout` 111, 114
`\setuplist` 195, 205
`\setupmargindata` 131
`\setuppagenumbering` 119
`\setuppapersize` 106
`\setupregister` 211
`\setupsectionblock` 185
`\setupspacing` 247
`\setupstretched` 248
`\setuptoptexts` 129
`\setuptype` 246
`\setuptyping` 246
`\setupurl` 227
`\setupuserpagenumber` 119
`\showbodyfontenvironment` 145
`\showcolor` 152, 153
`\showcolorcomponents` 153
`\showfont` 139
`\showframe` 113
`\showinstalledlanguages` 253
`\showlayouts` 113
`\showsetups` 113
`\showsymbolset` 240
`\sigma` 237
`\sl` 142
`\sla` 143
`\slanted` 142
`\slantedbold` 142
`\slb` 143
`\slc` 143
`\sld` 143
`\slx` 143
`\slxx` 143
`\smalcaps` 142
`\smallbodyfont` 145
`\smallbold` 145
`\smallbolditalic` 145
`\smallboldslanted` 145
`\smalllitalicbold` 145
`\smallslanted` 145
`\smallslantedbold` 145
`\somewhere` 221
`\space` 249
`\ss` 142, 236
`\ssa` 143
`\ssb` 143
`\ssc` 143
`\ssd` 143
`\ssx` 143
`\ssxx` 143
`\start` 76
`\startappendices` 185
`\startbackmatter` 185
`\startbodymatter` 185

`\startchapter` 164
`\startcolor` 153
`\startcomponent` 100
`\startenvironment` 98
`\startfrontmatter` 185
`\startMPpage` 109
`\startpagefigure` 109
`\startpart` 164
`\startproduct` 100
`\startproject` 101
`\startsection` 164
`\startsectionblockenvironment` 186
`\startsetups` 74
`\startsubject` 164
`\startsubsection` 164
`\startsubsubsection` 164
`\startsubsubsubject` 164
`\startsubsubsubsection` 164
`\startsubsubsubsubject` 164
`\startTEXpage` 109
`\starttext` 93
`\starttitle` 164
`\starttypescript` 158
`\starttyping` 245
`\stop` 76
`\stoptext` 93
`\stretched` 247
`\subject` 164
`\subsection` 164
`\subsubject` 164
`\subsubsection` 164
`\subsubsubject` 164
`\subsubsubsection` 164
`\subsubsubsubject` 164
`\support` 142
`\switchtobodyfont` 141
`\symbol` 241

t

`\TeX` 24, 25
`\tau` 237
`\tcedilla` 236
`\teletype` 142
`\tex` 246
`\textheight` 115
`\textreference` 218
`_` 249
`\textwidth` 115

`\tf` 142
`\tfa` 143
`\tfb` 143
`\tfc` 143
`\tfd` 143
`\tfx` 143
`\tfxx` 143
`\theta` 237
`\title` 164
`\translate` 255
`\tt` 142
`\tta` 143
`\ttb` 143
`\ttc` 143
`\ttd` 143
`\ttx` 143
`\ttxx` 143
`\tx` 143
`\txx` 143
`\typ` 246
`\type` 245

u

`\u` 235
`\uacute` 235
`\ubreve` 235
`\ucircumflex` 235
`\udiaeresis` 235
`\ugrave` 235
`\umacron` 235
`\upsilon` 237
`\usecolors` 152
`\usemodule` 253
`\usepath` 103
`\useregime` 86
`\userpagenumber` 121
`\usesymbols` 240
`\useURL` 226
`\utilde` 235

v

`\varepsilon` 237
`\varkappa` 237
`\varphi` 237
`\varpi` 237
`\varrho` 237
`\varsigma` 237
`\vartheta` 237

`\vbox` [123](#)

w

`\WORD` [243](#)

`\WORDS` [243](#)

`\Word` [243](#)

`\Words` [243](#)

`\word` [243](#)

`\writebetweenlist` [201](#)

`\writetolist` [200](#)

x

`\xi` [237](#)

z

`\zeta` [237](#)

Annexe B

Index

a

aide et ressources 33

c

compilation 41

composition 20

d

define 71

definestartstop 73

e

encode 84

espace 87

espace vide 87

f

fichier

composant 100

environnement 98

produit 100

projet 101

fichiers

chemin 103

fonts 156

OSFONTDIR 156

l

langages de balisage

balises 22

markup 22

m

Mark II 18

Mark IV 18

macro-structure 120

moteurs T_EX 24

mtxrun 156

p

page

dimension 106

préambule 44

r

rédaction 20

règles syntaxiques

commande 69

repertoire

chemin 103

s

saut de ligne 89

setupwhitespace 80

structure

chemin 103

composant 100

produit 100

projet 101

t

tabulation 87

tirets 90

trait d'union 90

Une courte (?) introduction à ConT_EXt Mark IV