

Une courte (?) introduction à ConT_EXt Mark IV

Une courte (?) introduction à ConT_EXt Mark IV

Version 1.6 [2 janvier 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](#).
- 2 janvier 2021: Version 1.6 : Corrections suggérées après une nouvelle lecture du document, à l'occasion de sa traduction en anglais

Table des matières

Préface	5
I Qu'est ce que ConT_EXt ? Comment travailler avec ? ...	13
1 ConT_EXt : une vue d'ensemble	14
1.1 Qu'est-ce que ConT _E Xt ?	14
1.2 La composition typographique de document	15
1.3 Les langages de balisage	17
1.4 T _E X et ses dérivés	18
1.5 ConT _E Xt	21
2 Notre premier fichier source	30
2.1 Préparation de l'expérience outils nécessaires	30
2.2 L'expérience elle-même	32
2.3 La structure de notre fichier d'exemple	38
2.4 Quelques détails supplémentaires sur la façon d'exécuter « context »	38
2.5 Traitement des erreurs	39
3 Les commandes et autres concepts fondamentaux de ConT_EXt	44
3.1 Les caractères réservés de ConT _E Xt	45
3.2 Les commandes à proprement parler	48
3.3 Périmètre des commandes	51
3.4 Options de fonctionnement des commandes	55
3.5 Résumé sur la syntaxe des commandes et des options, et sur l'utilisation des crochets et des accolades lors de leur appel.	58
3.6 La liste officielle des commandes ConT _E Xt	60
3.7 Définir de nouvelles commandes	61
3.8 Autres concepts fondamentaux	66
3.9 Méthode d'auto apprentissage pour ConT _E Xt	70
4 Source files and projects	73
4.1 Encoding source files	73

4.2	Characters in the source file(s) that ConT _E Xt treats in a special way .	76
4.3	Simple and multfile projects	79
4.4	Structure of the source file in simple projects	80
4.5	Multifile management in T _E X style	81
4.6	ConT _E Xt projects as such	84
 II Global aspects of the document		89
 III Particular issues		90
 Appendices		91

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 – 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.

Il existe une documentation considérable sur ConT_EXt, presque exclusivement en anglais. Par exemple, ce que l’on considère comme étant la distribution *officielle* de ConT_EXt – appelée « ConT_EXt Standalone »¹ – contient une documentation de quelques 180 fichiers PDF (la majorité en anglais, mais d’autres en néerlandais et en allemand) avec notamment des manuels, des exemples et des articles techniques ; et sur le [site web Pragma ADE](#) (la société qui a donné naissance à ConT_EXt) il y a (le jour où j’ai fait le décompte en mai 2020) 224 documents librement téléchargeables, dont la plupart sont distribués avec la « ConT_EXt Standalone » mais également quelques autres. Cependant, cette énorme documentation n’est pas particulièrement utile durant la phase d’apprentissage de ConT_EXt, car, en général, ces documents ne s’adressent pas à un lecteur désireux d’apprendre mais novice, qui ne connaît rien du système. Sur les 56 fichiers PDF que « ConT_EXt Standalone » appelle « manuels », un seul suppose que le lecteur ne connaît rien sur ConT_EXt. Il s’agit du document intitulé « [ConT_EXt Mark IV, an Excursion](#) »¹ ou en français « ConT_EXt Mark IV, une escapade ». Mais ce document, comme son nom l’indique, se limite à présenter le système et à expliquer comment faire certaines choses qui peuvent être faites avec ConT_EXt. Ce serait une bonne introduction s’il était suivi d’un manuel de référence un peu plus structuré et systématique. Mais un tel manuel n’existe pas et l’écart entre le document « [ConT_EXt Mark IV, an Excursion](#) » et le reste de la documentation est trop important.

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

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

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

Pour la liste, voir section ??.

les deux versions. Ce manque de différenciation se retrouve également dans la liste officielle des commandes « [ConT_EXt Commands](#) »¹ qui comprend, pour cha-

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

cune d'elles, l'ensemble des options de configuration possibles mais ne précise pas quelles commandes ne fonctionnent que dans l'une ou l'autre des versions.

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

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

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

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

In- Etant donné que ce qui est expliqué dans ce document provient essentiellement
certi- de mes propres conclusions, il est probable que, bien qu'ayant personnellement
tudes vérifié une grande partie de ce qui est exposé, certaines déclarations ou opinions

soient incorrectes ou non orthodoxes. Bien évidemment, j'apprécierai toute correction, toute nuance ou encore précisions que vous accepterez de me faire parvenir à joaquin@ataz.org. Pour limiter les occasions d'erreur, j'ai essayé de ne pas aborder les sujets sur lesquels je n'ai pas trouvé d'informations ou que je n'ai pas pu (ou voulu) vérifier personnellement ; parce que parfois le résultat de mon test n'était pas concluant, d'autres fois parce que je n'ai pas toujours tout essayé : le nombre de commandes et d'options de 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 conjecture¹. 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.



Pu- Autre aspect, cette introduction a été écrite pour un lecteur qui ne connaît rien
blic à T_EX et ConT_EXt, bien que j'espère qu'elle pourra également être utile à ceux
visé 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.

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

tion

- **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 ?? et ?? sont consacrés à ces deux questions.

- * Le Chapitre ?? 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 ?? 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 ?? et ?? se concentrent sur la structure du document et les outils que ConTeXt met à la disposition de l'auteur pour la rédaction de documents bien structurés. Le Chapitre ?? se concentre sur la structure elle-même (divisions structurelles du document) et le Chapitre ?? sur la valorisation de cette structure dans une table des matières.
- Enfin, le Chapitre ?? se concentre sur l'aspect clé de l'utilisation de références par un document, références vers d'autres points du même document (références internes) mais aussi vers des documents externes (références externes). Dans cette dernière situation, nous n'aborderons que le cas de références impliquant un *liens* vers le document externe. Ce chapitre explique l'utilisation de l'outil de ConTeXt pour la gestion de ces références qui permet des *sauts* entre différentes zones d'informations, internes ou externes, et rend ainsi notre document *interactif*.

Ces chapitres n'ont pas besoin d'être lus dans un ordre particulier, sauf peut-être le Chapitre ??, qui est peut-être plus facile à comprendre si vous avez d'abord lu le Chapitre ??. En tout cas, j'ai essayé de faire en sorte que lorsqu'une question se pose dans un chapitre ou une section, alors qu'elle est traitée ailleurs dans ce document, le texte mentionne ce fait et propose un hyperlien vers le point où cette question est traitée. Je ne suis cependant pas en mesure de garantir que ce sera toujours le cas.

- Enfin, **la troisième partie** (Chapitre ?? et suivants) se concentre sur des aspects plus concrets, plus spécialisés. Non seulement les chapitres sont maintenant indépendants les uns des autres, mais également les sections les unes des autres au sein d'un même chapitre (sauf, peut-être, dans le dernier chapitre). Étant donné la grande quantité de fonctionnalités proposées par ConTeXt cette troisième partie pourrait être très vaste ; mais comme je devine qu'en arrivant à ce stade le lecteur sera capable de plonger lui-même dans la documentation de ConTeXt je n'ai considéré que les chapitres suivants :
 - Les chapitres ?? et ?? traitent de ce que l'on pourrait appeler les *éléments clés* de tout document texte : Le texte est constitué de caractères, qui

forment des mots, qui sont regroupés en lignes, qui à leur tour forment des paragraphes, qui sont séparés les uns des autres par un espace vertical... Il est évident que toutes ces questions auraient pu être incluses dans un seul chapitre. Mais comme cela aurait été trop long, j'ai divisé cette question en deux chapitres, l'un traitant des caractères, des mots et des espaces horizontaux, et l'autre traitant des lignes, des paragraphes et des espaces verticaux.

- Le Chapitre ?? est une sorte de *pot-pourri* qui traite des éléments et constructions qui sont communs à de nombreux documents, principalement s'ils sont académiques ou scientifico-techniques : notes de bas de page, listes structurées, descriptions, énumérations, etc.
- Enfin, le Chapitre ?? se concentre sur les objets flottants insérés dans un document et en particulier les deux cas les plus répandus : les images et les tableaux.
- Cette introduction à ConT_EXt se termine par trois **Annexes**. L'une concerne l'installation de ConT_EXt sur un ordinateur, une deuxième annexe présentent plusieurs dizaines de commandes qui permettent de générer divers symboles (principalement, mais pas uniquement, pour un usage mathématique), et une troisième annexe rassemble les commandes ConT_EXt expliquées ou mentionnées tout au long de ce texte sous la forme d'une liste alphabétique.

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

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

Joaquín Ataz-López
Été 2020

I

Qu'est ce que ConT_EXt ?

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; 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.

- ConTeXt n'est pas un programme de traitement de texte ou d'édition de texte, mais un ensemble d'outils conçus pour *transformer* un texte que nous avons précédemment écrit avec notre éditeur de texte préféré. Par conséquent, lorsque nous travaillons avec ConTeXt :
 - Nous commençons par écrire un ou plusieurs fichiers texte avec n'importe quel éditeur de texte.
 - Dans ces fichiers, en plus du texte qui constitue le contenu réel du document, il y a une série d'instructions, instructions qui indiquent à ConTeXt à quoi doit ressembler le document final généré à partir des fichiers texte originaux. L'ensemble complet des instructions ConTeXt constitue en fait un *langage* ; et puisque ce langage permet de *programmer* la transformation typographique d'un texte, on peut dire que ConTeXt est un *langage de programmation typographique*.
 - Une fois les fichiers sources écrits, ils seront traités par un programme (également appelé « context »¹), qui, à partir de ceux-ci, générera un fichier PDF prêt à être envoyé à une imprimante ou à être affiché à l'écran.
- Dans ConTeXt, nous devons donc faire la différence entre le document que nous rédigeons et le document que ConTeXt génère. Pour éviter tout doute, dans cette introduction, j'appellerai *fichier source* le document texte qui contient les instructions de formatage, et *document final* le fichier PDF généré par ConTeXt à partir du fichier source.

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

1.2 La composition typographique de document

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

L'écriture du document c'est sa rédaction, qui est effectuée par l'auteur, qui décide de son contenu et de sa structure. Le document créé directement par l'auteur, tel qu'il l'a écrit, s'appelle un *manuscrit*. Le manuscrit, par sa nature même, n'est accessible qu'à l'auteur et aux personnes à qui l'auteur permet de le lire. Sa diffusion au-delà de ce cercle intime nécessite que le manuscrit soit *publié*. De nos jours, publier quelque chose — au sens étymologique de «rendre accessible au public» — est aussi simple que de le mettre sur l'internet, à la disposition de quiconque peut le localiser et souhaite le lire. Mais jusqu'à une date relativement récente, l'édition était un processus qui impliquait des coûts, dépendait de certains

ConTeXt désigne donc à la fois un langage et un programme (ainsi qu'un ensemble d'autres outils formant le système complet). Par conséquent, dans un texte comme cette introduction, se pose le problème de devoir parfois faire la distinction entre les deux aspects. J'ai donc adopté la convention typographique consistant à écrire «ConTeXt» avec son logo (ConTeXt) lorsque je veux me référer exclusivement à la langue, ou indistinctement à la langue et au programme. Et lorsque je veux me référer exclusivement au programme j'écrirai « context » tout en minuscules et avec une police de caractères à espacement constant, typique des terminaux d'ordinateur et des machines à écrire, que j'utiliserai aussi pour les exemples et pour faire référence aux instructions du langage.

professionnels spécialisés dans ce domaine et n'était accessible qu'aux manuscrits qui, en raison de leur contenu ou de leur auteur, étaient considérés comme particulièrement intéressants. Et aujourd'hui encore, nous avons tendance à réserver le mot *publication* au type de *publication professionnelle* par lequel le manuscrit subit une série de transformations de son apparence visant à améliorer la *lisibilité du document*. C'est cette série de transformations qui est appelée *composition* ou encore *composition typographique*.

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

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

De nos jours, grâce à l'informatique, il est plus facile pour l'auteur lui-même de définir la composition jusqu'aux moindres détails. Mais pour autant les qualités d'un bon auteur ne sont pas les mêmes que celles d'un bon compositeur. L'auteur doit avoir une bonne connaissance du sujet traité, savoir le structurer, l'exposer avec clarté, avec créativité, avec un rythme. etc. Le compositeur typographe doit avoir une bonne connaissance de l'environnement graphique et conceptuel à sa disposition, un goût de l'esthétique pour les utiliser harmonieusement, de façon cohérente avec le sujet traité, avec les tendances du moment.

Avec un bon logiciel de traitement de texte ¹, 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

¹ 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.

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.

¹ 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).

En ce qui concerne les balises *typographiques*, qui indiquent l'apparence que doit avoir un texte, il en existe deux types, que nous pourrions distinguer comme d'un côté les *balises purement typographique* (ou encore graphiques) et de l'autre les *balises sémantiques* (ou encore conceptuelles, logiques). Les balises purement typographiques se limitent à indiquer précisément quelles ressources typographiques doivent être utilisées pour afficher un certain texte ; par exemple, lorsque nous indiquons qu'un certain texte doit être en gras ou en italique, de telle ou telle couleur. Le balisage sémantique, quant à lui, indique la fonction d'un texte donné dans l'ensemble du document, par exemple lorsque nous indiquons qu'il s'agit d'un titre, d'un sous-titre, d'une citation. En général, les documents qui utilisent de préférence ce deuxième type de balisage sont plus cohérents et plus faciles à composer, car la différence entre la paternité et la composition y est à nouveau claire : l'auteur indique que cette ligne est un titre, ou que ce fragment est un avertissement, ou une citation ; et le compositeur décide comment mettre en valeur typographiquement tous les titres, avertissements ou citations ; ainsi, d'une part, la cohérence est garantie, puisque tous les fragments remplissant la même fonction auront la même apparence, et, d'autre part, on gagne du temps, puisque le format de chaque type de fragment ne doit être indiqué qu'une seule fois.

1.4 T_EX et ses dérivés

T_EX a été développé à la fin des années 1970 par DONALD E. KNUTH, professeur de théorie de la programmation à l'université de Stanford, qui l'a utilisé pour composer ses propres publications et ainsi que pour donner un exemple de *programmation littéraire*, une approche de la programmation où le code source du logiciel est systématique commenté et documenté. Avec T_EX, KNUTH a également développé un langage de programmation supplémentaire appelé METAFONT, pour la conception de caractères typographiques, avec lequel il a créé une police qu'il a nommée *Computer Modern*, qui, en plus des caractères habituels de toute police, comprenait également un ensemble complet de «glyphes» ¹ pour l'écriture

des mathématiques. Il a ajouté à tout cela quelques utilitaires supplémentaires et c'est ainsi qu'est né le système de composition appelé \TeX , qui, pour sa puissance, la qualité de ses résultats, sa flexibilité d'utilisation et ses vastes possibilités, est considéré comme l'un des meilleurs systèmes informatiques pour la composition de textes. Il a été pensé pour des textes dans lesquels il y avait beaucoup de mathématiques, mais on a vite vu que les possibilités du système le rendaient adapté à tous les types de textes.

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

Le nom \TeX vient de la racine du mot grec $\tau\acute{\epsilon}\chi\upsilon\eta$, écrit en lettres capitales (ΤΕΧΝΗ). 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 se écrit à écrire :

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

Pour rendre plus facile l'écriture d'un tel logo, Knuth a inclus dans une instruction qui l'inscrit dans le document final : `\TeX`.

- Dans un texte non formaté (tel qu'un e-mail ou un fichier texte), le « T » et le « X » sont en majuscules, et le « e » du milieu est en minuscules ; par exemple : « `TeX` ».

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

1.4.1 Les moteurs T_EX

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

La question du choix du moteur T_EX à utiliser fait l'objet d'un débat animé dans l'univers T_EX. Je ne m'y attarderai pas ici, car ConT_EXt Mark IV ne fonctionne qu'avec LuaT_EX. En fait, dans le monde de ConT_EXt la discussion sur les moteurs devient une discussion sur l'utilisation de Mark II (qui fonctionne avec pdfT_EX et X_YT_EX) ou Mark IV (qui fonctionne avec LuaT_EX).

1.4.2 Les formats dérivés de T_EX

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

Pour cette raison, K_NUTH a rendu T_EX extensible, c'est-à-dire disposant d'un mécanisme permettant de définir des instructions de plus haut niveau, dans un langage plus facilement compréhensibles par les êtres humains. Ces instructions, qui au moment de l'exécution sont décomposées en instructions élémentaires, sont appelées *macros*. Par exemple, l'instruction T_EX qui imprime votre logo ($\text{\textcolor{violet}{T}\text{\textcolor{violet}{e}}\text{\textcolor{violet}{X}}$), 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 « `\TeX` » dont l'exécution effectue l'ensemble des opérations typographiques nécessaires à l'impression du logo.

La différence entre les *macros* et les *primitives* n'est vraiment importante que du point de vue du développeur de T_EX. 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 T_EX un outil si puissant. En fait, KNUTH lui-même a conçu environ 600 macros qui, avec les 300 primitives, constituent le format appelé « Plain T_EX ». Il est assez courant de confondre T_EX lui-même avec Plain T_EX et, en fait, presque tout ce qui est dit ou écrit sur T_EX, se réfère en fait à Plain T_EX. Les livres qui prétendent être sur T_EX (y compris le livre fondateur « *The T_EXBook* »), font en fait référence à Plain T_EX ; et ceux qui pensent qu'ils manipulent directement T_EX manipulent en fait Plain T_EX.

Plain T_EX est ce que l'on appelle dans la terminologie T_EX 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 T_EX, d'autres formats ont été développés au fil du temps, notamment L^AT_EX un vaste ensemble de macros pour T_EX conçu en 1985 par LESLIE LAMPORT, qui est probablement le dérivé de T_EX le plus utilisé dans le monde universitaire, technologique et mathématique. ConT_EXt est (ou a commencé à être), de même que L^AT_EX un format dérivé de T_EX.

Normalement, ces *formats* sont accompagnés d'un programme qui charge dans la mémoire de l'ordinateur les macros qui les composent avant d'appeler « `tex` » (ou l'un des autres moteurs précédemment listés) pour traiter le fichier source. Mais bien que tous ces formats exécutent finalement T_EX, 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 T_EX, mais différents de T_EX, 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, officiellement 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 a 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 a 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, ConTeXt est né au sein d'une entreprise dédiée à la composition de documents. Par conséquent, loin d'essayer d'isoler l'auteur des détails de la composition, ce qu'il tente de faire, c'est de lui donner un contrôle absolu et complet sur ceux-ci. Pour ce faire, ConTeXt fournit une interface homogène et cohérente qui reste beaucoup plus proche de l'esprit original T_EX que L^AT_EX.

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

Contrairement à la simplicité du noyau de L^AT_EX et son extensibilité par le biais de paquets, ConTeXt 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 ConTeXt permet de faire presque tout et il est garanti qu'il n'y aura pas d'incompatibilités entre les différentes commandes, il n'y a pas besoin de rechercher les extensions dont vous avez besoin (elles sont déjà présentes), et la syntaxe du langage est homogène entre les différentes commandes.

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

Je ne pense pas que l'une de ces deux *philosophies* puisse être considérée comme préférable à l'autre. La réponse dépend plutôt du profil de l'utilisateur et de ce

qu'il souhaite. Si l'utilisateur ne veut pas s'occuper de questions typographiques, mais simplement produire des documents standardisés de très haute qualité, il serait probablement préférable pour lui d'opter pour un système comme 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; 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 $\text{T}_{\text{E}}\text{X}$ 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 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ que pour ConTeXt. GNU Emacs en combinaison avec AucTeX peut être une bonne option si l'on ne sait pas quel éditeur choisir ; tous deux sont des programmes à code source ouvert, et ils sont disponibles pour tous les systèmes d'exploitation. En fait, dire que GNU Emacs est un *logiciel libre* est un euphémisme, car ce programme incarne mieux que tout autre l'esprit de ce qu'est et signifie le *logiciel libre*. Après tout, son principal développeur était RICHARD STALLMAN, fondateur et idéologue du projet GNU et de la *Free Software Foundation*.

En plus de GNU Emacs + AucTeX, Scite et TexWorks sont d'autres bonnes options si vous ne savez pas quel éditeur choisir. Le premier, bien qu'il s'agisse d'un éditeur à usage généraliste, non conçu spécifiquement pour travailler avec des fichiers ConTeXt, est facilement personnalisable et, comme c'est l'éditeur généralement utilisé par les développeurs de ConTeXt « ConTeXt Standalone » contient les fichiers de configuration de cet éditeur conçus et utilisés par HANS HAGEN lui-même. TexWorks est, quant à lui, un éditeur de texte rapide, spécialisé dans le traitement des fichiers $\text{T}_{\text{E}}\text{X}$ et de ses langages dérivés. Il est assez facile à configurer pour fonctionner avec ConTeXt et « ConTeXt Standalone » prévoit également de fournir des fichiers configurations.

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

2. **Une distribution ConTeXt** pour traiter notre fichier de test. S'il existe déjà une installation $\text{T}_{\text{E}}\text{X}$ (ou $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$) sur votre système, il est possible qu'une version de ConTeXt soit déjà installée. Pour le vérifier, il suffit d'ouvrir un terminal et de taper dans celui-ci

```
$ context --version
```

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

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

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

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.

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.

Regardons le contenu de notre premier fichier source pour ConTeXt

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

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

Sur la 9^{ème} ligne de notre exemple (je ne compte que les lignes qui ont du texte) se trouve la commande importante `\starttext` : elle indique à ConTeXt que le contenu du document commence à partir de cet endroit; et à la dernière ligne de notre exemple, nous voyons la commande `\stoptext` qui indique la fin du contenu. Tout ce qui suit cette dernière commande ne sera pas traité. Ce sont deux commandes très importantes sur lesquelles je reviendrai très bientôt. Entre les deux se trouve donc le contenu à proprement parler de notre document qui, dans notre exemple, consiste en la première strophe de la chanson "« La maison sur le port », dont les paroles sont de AMALIA RODRIGUES, et qui a été reprise notamment par SANSEVERINO. Je l'ai écrit en prose afin de mieux observer le formatage des paragraphes effectué par ConTeXt.

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 avions appelé au fichier source, par exemple, « `la-maison-sur-le-port.mkiv` » (ce que je fais habituellement pour savoir que ce fichier est écrit pour Mark IV), il aurait fallu indiquer expressément l'extension du fichier à compiler en tapant « `context la-maison-sur-le-port.mkiv` ».

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

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

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

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

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

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

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

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

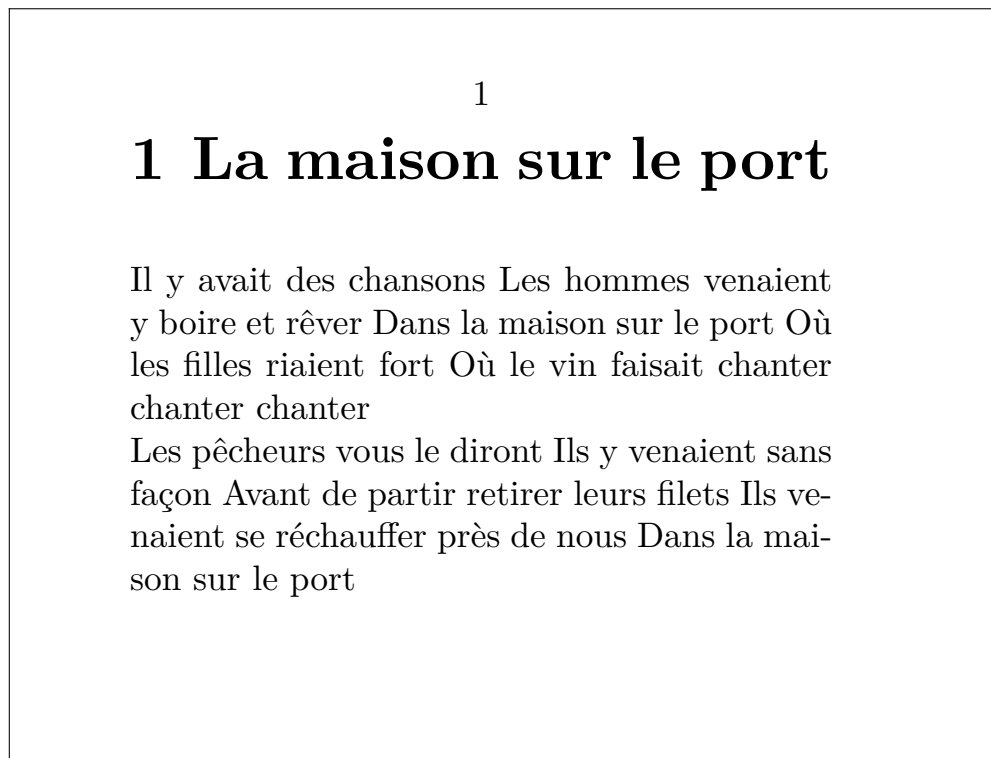


Figure 2.1 La maison sur le port

La [figure 2.1](#) montre le contenu de « la-maison-sur-le-port.pdf ». Nous pouvons voir que ConT_EXt a numéroté la page, numéroté le chapitre et écrit le texte dans la police indiquée. Il a également réparti le mot « venaient » entre la sixième et la septième ligne, ainsi que le mot « maison » la septième et la huitième ligne. ConT_EXt, par défaut, active la césure (division syllabique) des mots afin de répartir les blancs (les espaces vides entre les mots) de façon la plus homogène possible. C'est pourquoi il est si important d'informer ConT_EXt de la langue du document, car les modèles de césure varient selon la langue. Dans notre exemple, c'est l'objectif de la première commande du fichier source (`\mainlanguage[fr]`).

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

2.3 La structure de notre fichier d'exemple

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

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

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

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

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

```
$ context --help
```

Parmi les options les plus intéressantes, citons les suivantes :

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

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

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

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

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

version indique la version de ConTeXt.

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

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

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

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

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

2.5 Traitement des erreurs

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

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

Que faire lorsque « context » nous avertit, pendant la compilation, qu'une erreur s'est produite ? La première chose est, évidemment, d'identifier quelle est l'erreur. Pour ce faire, nous devons 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.

Par exemple, si dans notre fichier de test, « la-maison-sur-le-port.tex », par erreur, au lieu de `\starttext` nous avons écrit `\startext` (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


```

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

```

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 `\startext`. Nous le lisons attentivement et, avec de l'attention et de l'expérience, nous nous rendons compte que nous avons écrit « startext » et non « startttext » (avec un double « t »).

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

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

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

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

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

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

Chapitre 3

Les commandes et autres concepts fondamentaux de ConT_EXt

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

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

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

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

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

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).¹
- % 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

¹ 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.

d'une certaine manière, comment tel ou tel effet graphique a été obtenu, garder un rappel d'une idée à compléter ou à réviser, d'une illustration à construire.

Il peut également être utilisé pour aider à localiser une erreur dans le fichier source, puisqu'en commentant une ligne, nous l'excluons de la compilation, et pouvons voir si elle est à l'origine de l'erreur de compilation. Le commentaire peut aussi être utilisé pour stocker deux versions différentes d'une même macro, et ainsi obtenir des résultats différents après la compilation ; ou pour empêcher la compilation d'un extrait dont nous ne sommes pas sûrs, mais sans le supprimer du fichier source au cas où nous voudrions y revenir plus tard ; ou pour partager des commentaires lors de l'édition en mode collaboratif d'un document... etc.

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

- { Ce caractère ouvre un groupe. Les groupes sont des blocs de texte auxquels on souhaite appliquer certaines effet ou affecter certaines caractéristiques. Nous en parlerons dans la section ??.
- } Ce caractère 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, ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 Écriture 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 ConT_EXt 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 (`_` `^` `&`) : s'ils sont utilisés en dehors de cet environnement, ils sont traités comme des caractères normaux.

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



3.2 Les commandes à proprement parler

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

- a. **Symboles de contrôle.** Un symbole de contrôle commence par la séquence d'échappement (« `\` ») et consiste exclusivement en un caractère autre qu'une lettre, comme par exemple « `\,` », « `\1` », « `\'` » ou « `\%` ». Tout caractère ou symbole qui n'est pas une lettre au sens strict du terme peut être un symbole de contrôle, y compris les chiffres, les signes de ponctuation, les symboles et même un espace vide. Dans ce document, pour représenter un espace vide

(espace blanc) lorsque sa présence doit être soulignée, le symbole que j'utilise est `_`. En fait, « `_` » (une barre oblique inversée suivie d'un espace blanc) est un symbole de contrôle couramment utilisé, comme nous pourrions bientôt le constater.

Un espace vide ou blanc est un caractère « invisible », ce qui pose un problème dans un document comme celui-ci, où il faut parfois préciser clairement ce qui doit être écrit dans un fichier source. Knuth était déjà conscient de ce problème et, dans son « The T_EXBook », il a pris l'habitude de représenter les espaces vides importants par le symbole « `_` ». Ainsi, par exemple, si nous voulions montrer que deux mots du fichier source doivent être séparés par deux espaces vides, nous écrivions « `word1_word2` ».

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

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

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 ConT_EXt dans un cadre de fond gris. Le résultat de la compilation est présenté dans un cadre de fond de couleur jaune foncé.



Lorsque ConT_EXt 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 ConT_EXt 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 ConT_EXt 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 : ¹

```

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 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.

```

Notez comment, dans le premier cas, le mot « T_EX » 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 ConT_EXt 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 ConT_EXt (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 ConT_EXt à savoir que le nom de la commande qui le précède est terminé. On peut aussi créer un groupe autour de la commande en question, par exemple en écrivant « `{\TeX}` ». Dans les deux cas, le résultat sera que la première « non-lettre » après `\TeX` n'est pas un espace vide.
- Le symbole de contrôle « `_` » (une barre oblique inverse suivie d'un espace vide, voir la note sur [page 49](#)) utilisé à la cinquième ligne de l'exemple. L'effet de ce symbole de contrôle est d'insérer un espace blanc dans le document final. Pour bien comprendre la logique de 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 ConT_EXt 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_NU_TH 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

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 ² avec un argument.

¹ Pas toujours, cela dépend de l'environnement en question et de la situation dans le reste du document. ConT_EXt diffère de L^AT_EX à cet égard qui est beaucoup plus stricte.
² test

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 ConT_EXt. 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 ConT_EXt 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.¹ 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 fermeture ² d'environnement va impliquer une mauvaise interprétation par ConT_EXt qui soulignera le passage qu'il considère comme éronné et non pas

le manque de fermeture d’environnement, ce qui signifie que le fichier « .log » ne nous est pas d’une grande aide pour trouver où se situe le problème.

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

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

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

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

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



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

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

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

```
\framed{Tweedledum}
```

et celui d'une version personnalisée :

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

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

```

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

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

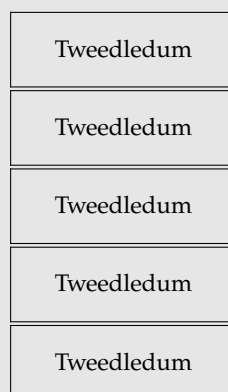
\framed
[width=3cm, height=1cm]
{Tweedledum}

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

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

\

```



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

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

3.4.2 Les commandes qui configurent comment d’autres commandes fonctionnent (`\setupQuelqueChose`)

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 ConT_EXt 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 ConT_EXt 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 ConT_EXt son extrême puissance et flexibilité. Si nous examinons en détail ce que fait la commande `\defineSomething`, nous constatons que :

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

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

```
\defineframed
  [MonCadre]

\setupframed
  [MonCadre]
  [MaConfigurationPourCadre]
```

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

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

En résumant ce que nous avons vu jusqu’à présent, nous voyons que dans ConT_EXt

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

Lorsque la commande ne doit agir que sur une seule lettre, comme c’est le cas, par exemple, de la commande `\buildtextcedilla` (pour donner un exemple – le « ç »

si souvent utilisée en catalan), les accolades autour de l'argument peuvent être omises : la commande s'appliquera au premier caractère qui n'est pas un espace blanc.

- Certains arguments sont facultatifs, auquel cas nous pouvons les omettre. Mais ce que nous ne pouvons jamais faire, c'est changer l'ordre des arguments que la commande attend.

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

Ils peuvent prendre trois forme différentes :

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

Enfin, il n'arrive jamais avec ConT_EXt 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 texe pour cette section
```

```
FONCTION
```

```
\stopsection
```

1 mon titre[5]

avec crochets] Du texte pour cette section NE FONCTIONNERA PAS

2 mon titre[5] avec crochets

Du texe 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) + setupCOMMANDE + 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, (`\definblank`).

« `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), ConT_EXt 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{0 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

```
\define\xvec{$(x_1,\ldots,x_n)$}
```

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

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

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

où

- **NbrArguments** 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, 16 mai 2021
Chère Madame,

Anne Smith
Consultant

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

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

Marseille, 16 mai 2021
Chère Madame Dupond,

Anne Smith
Consultant

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

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

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

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

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

Anne Smith
Consultant

Marseille, 16 mai 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 `\definestartstop` dont la syntaxe est la suivante :

```
\definestartstop[Nom] [Configuration]
```

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



où

- **Nom** est le nom que portera le nouvel environnement.
- **Configuration** nous permet de configurer le comportement du nouvel environnement. Nous disposons des valeurs suivantes avec lesquelles nous pouvons le configurer :
 - **before** : Commandes à exécuter avant d'entrer dans l'environnement.
 - **after** : Commandes à exécuter après avoir quitté l'environnement.
 - **style** : Style que doit avoir le texte du nouvel environnement.

- `setup` : Ensemble de commandes créées avec `\startsetups ... \stopsetups`. Cette commande et son utilisation ne sont pas expliquées dans cette introduction.
- `color` : Couleur à appliquer au texte
- `inbetween`, `left`, `right` : Options non documentées que je n’ai pas réussi à faire fonctionner. D’après les tests que j’ai effectués, indiquant une certaine valeur pour ces options, je ne vois aucun changement dans l’environnement. Il est possible que l’impact ne concerne pas l’environnement mais la commande qui semble créer avec `\Nom`. Une piste sur [la liste de diffusions NTG-context](#).



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

```
\definestartstop
[TextWithBar]
[before=\bgroup\startmarginrule\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 ConT_EXt 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 ConT_EXt. 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.

¹ La notion de *boîte* est également une notion centrale de ConT_EXt 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 ConT_EXt 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*¹. 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 ConT_EXt modifie temporairement son mode de fonctionnement normal, le moyen le plus efficace est de créer un groupe et, au sein de celui-ci, de modifier ce fonctionnement. Ainsi, lorsque nous quitterons le groupe, toutes les valeurs et tous les formats antérieurs à celui-ci seront restaurés. Nous en avons déjà vu quelques exemples en mentionnant des commandes comme `\it`, `\bf`, `\sc`, etc. Mais cela ne se produit pas seulement avec les commandes de formatage : le groupe isole en quelque sorte son contenu, de sorte que toute modification de l'une des nombreuses variables internes que ConT_EXt gère en permanence, ne restera effective que tant que nous nous trouvons dans le groupe dans lequel cette modification a eu lieu. De même, une commande définie au sein d'un groupe ne sera pas connue en dehors de celui-ci.

Ainsi, si nous traitons l'exemple suivant

<pre> \define\A{B} \A { \define\A{C} \A } \A </pre>	<pre> B C B </pre>
---------------------------------------------------------	--------------------

nous voyons que la première fois que nous exécutons la commande `\A`, le résultat correspond à celui de sa définition initiale («B»). Ensuite, nous avons créé un groupe et redéfini la commande `\A` au sein de celui-ci. Si nous l'exécutons maintenant au sein du groupe, la commande nous donnera la nouvelle définition («C»)

dans notre exemple), mais lorsque nous quittons le groupe dans lequel la commande `\A` a été redéfinie, si nous l'exécutons à nouveau, elle tapera «B» une fois de plus. La définition faite au sein du groupe est « oubliée » une fois que nous l'avons quitté.

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

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

3.8.2 Dimensions

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

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

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

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

Tableau 3.2 Unités de mesure dans ConT_EXt

contexte d'utilisation puisque tout est mis en cohérence avec la taille de la police de caractère. C'est pourquoi, en général, elle est recommandée.

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

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

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

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

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



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

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

il est vraiment difficile d'obtenir une certaine cohérence dans les caractéristiques typographiques.

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

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

Par exemple :

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

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

3.9 Méthode d'auto apprentissage pour ConT_EXt

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

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

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

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

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

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

Pour savoir si ces commandes existent ou non, consultez la liste officielle des commandes (voir section ??), qui nous informera également des options de configuration que ces commandes prennent en charge. Bien qu'à première vue, les noms de ces options puissent sembler cryptiques, nous verrons rapidement que certaines options sont répétées dans de nombreuses commandes et qu'elles fonctionnent de la même manière ou de manière très similaire dans toutes ces commandes. Si nous avons des doutes sur ce que fait une option, ou sur son fonctionnement, il suffira de générer un document et de le tester. Nous pouvons également consulter l'abondante documentation de ConT_EXt. Comme il est courant dans le monde des logiciels libres, « ConT_EXt Standalone » inclut les sources de presque toute sa documentation dans la distribution. Un utilitaire comme « `grep` » (pour les systèmes GNU Linux) peut nous aider à rechercher si la commande ou l'option sur laquelle nous avons des doutes est utilisée dans l'un de ces fichiers sources afin d'avoir un exemple sous la main.

C'est ainsi que l'apprentissage de ConT_EXt a été conçu : l'introduction explique en détail les cinq (en réalité quatre) aspects que j'ai mis en évidence, et bien d'autres

encore : au fil de la lecture, une image claire de la séquence se formera dans notre esprit : *une commande pour exécuter la tâche – une seconde commande pour configurer le comportement de la première – une troisième commande pour créer un commande similaire à la première à partir d'un clône*. Nous apprendrons également certaines des principales structures de ConT_EXt, et nous saurons à quoi elles servent.

Chapitre 4

Source files and projects

Table of Contents: 4.1 Encoding source files; 4.2 Characters in the source file(s) that ConTeXt treats in a special way; 4.2.1 Blank spaces (white space) and tabs; 4.2.2 Line breaks; 4.2.3 Rules/dashes; 4.3 Simple and multifile projects; 4.4 Structure of the source file in simple projects; 4.5 Multifile management in TeX style; 4.5.1 The `\input` command; 4.5.2 `\ReadFile` and `\readfile`; 4.6 ConTeXt projects as such; 4.6.1 *Environment* files; 4.6.2 Components and products; 4.6.3 Projects as such; 4.6.4 Common aspects of environments, components, products and projects;

As we already know, when working with ConTeXt we always start with a text file in which, along with the contents of the final document, a number of instructions are included, telling ConTeXt about the transformations it must apply to generate our final correctly formatted document in PDF from the source file.

Thinking of the readers who until now have only known how to work with word processors, I think it is worth spending some time with the source file itself. Or rather, source files, since there are times where there is only one source file and others when we use a number of source files to arrive at the final document. In this last instance we can talk about « multifile projects ».

4.1 Encoding source files

The source file(s), need to be text files. In computer terminology, this is the name given to a file containing only human-readable text which does not include binary code. These files are also called *simple text* or *plain text* files.

Since internally, computer systems only process binary numbers, a text file is really made up of *numbers* that represent *characters*. A *table* is used to connect a number with a character. For text files, there are several possible tables. The term *text file encoding* refers to the specific character-matching table that a particular text file uses.

The existence of different encoding tables for text files is a consequence of the history of computer science itself. In the early stages of development, when the memory and storage capacity of computer devices was scarce, it was decided to use a table called ASCII (standing for « *American Standard Code for Information Interchange* ») that only allowed 128 characters and was established in 1963 by the US Standards Committee. It is obvious that 128 characters is not enough to represent all the characters and symbols used in

all the languages of the world; but it was more than enough to represent English which is, of all Western languages, the one that has fewer characters, because it does not use any diacritics (accents and other marks above or below or through other letters). The advantage of using ASCII was that text files would take up very little space, as 127 (the highest number in the table) can be represented by a 7-digit binary number, and the first computers used the byte as a unit for measuring memory, an 8-digit binary number. Any character in the table would fit into a single byte. Since the byte has 8 digits and ASCII used only 7 digits, there was even space left to add some other characters to represent other languages.

But when the use of computers expanded, the inadequacy of ASCII became apparent and it became necessary to develop *alternative* tables that included characters not known to the English alphabet such as the Spanish «ñ», accented vowels, the Catalan or French «ç», etc. On the other hand, there was no initial agreement as to what these *alternative tables* of ASCII should be, so different specialised computer companies gradually tackled the problem on their own. Therefore, not only were specific tables created for different languages or groups of languages, but also different tables according to the company that had created them (Microsoft, Apple, IBM, etc.).

It was only with the increase in computer memory and the lower cost of storage devices and the corresponding increase in capacity that the idea of creating a single table that could be used for all languages arose. But, once again, it was not actually a single table containing all the characters that was created, but a standard encoding (called Unicode) along with different ways of representing it (UTF-8, UTF-16, UTF-32, etc.) Of all these systems, the one that has ended up becoming the de facto standard is UTF-8, which makes it possible to represent practically any living language, and many already extinct languages, as well as numerous additional symbols, all using numbers of variable length (between 1 and 4 bytes), which, in turn, helps to optimise the size of text files. This size has not increased *too much* compared to files using pure ASCII.

Up until X₃T_EX appeared, systems based on T_EX – which was also born in the US and therefore has English as its native language – assumed the encoding was in pure ASCII; so that to use a different encoding, you had to indicate this somehow in the source file.

ConT_EXt Mark IV assumes that the encoding will be UTF-8. However, in less up-to-date computer systems, a different encoding may still be used by default. I am not very sure about the default encoding that Windows uses, given that Microsoft's strategy for reaching out to the wider public consists in hiding the complexity (but even though hidden, it does not mean it has disappeared!). There is not much information available (or I have not been able to find it) regarding the encoding system it uses by default.

In any case, whatever the default encoding, any text editor allows you to save the file in the desired encoding. The source files intended to be processed by ConT_EXt Mark IV must be saved in UTF-8, unless, of course, there is a very good reason for using a different encoding (although I cannot think what this reason might be).

If we want to write a file written in another encoding (perhaps an old file) we can

- a. Convert the file to UTF-8, the recommended option, and there are various tools for doing this; in Linux, for example, the commands `iconv` or `recode`.
- b. Tell ConT_EXt in the source file that the encoding is not UTF-8. To do this we need to use the command `\enableregime`, the syntax of which is:

`\enableregime[Encoding]`

where *Encoding* refers to the name by which ConT_EXt knows the actual encoding of the file in question. In the [table 4.1](#) you will find the various encodings and the names that ConT_EXt knows them by.

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

Tableau 4.1 Main encodings in ConT_EXt

ConT_EXt Mk IV strongly recommends the use of UTF-8. I agree with this recommendation. From here on in this introduction, we can assume that the encoding is always UTF-8.

Along with `\enableregime` ConT_EXt includes the command `\useregime` which allows us to use the code for one or other encodings as an argument. I have found no information about this command nor how it differs from `\enableregime`, only some examples of its use. I suspect that `\useregime` is designed for complex projects that use many source files, with the expectation that not all of them will have the same coding. But it is only a guess.



4.2 Characters in the source file(s) that ConTeXt treats in a special way

Special characters is the name I will give to a group of characters that are different from *reserved characters*. As seen in [section 3.1](#), they are ones that have a special meaning for ConTeXt and so cannot be used directly as characters in the source file. Along with these there is another group of characters that, although treated as such by ConTeXt when it finds them in the source file, it does treat them with special rules. This group includes blank spaces (white space), tabs, line breaks and hyphens.

4.2.1 Blank spaces (white space) and tabs

Tabs and blank spaces are treated the same in the source file for all intents and purposes. A tab character (the Tab key on the keyboard) will be transformed into white space by ConTeXt. And blank spaces are absorbed into any other blank space (or tab) immediately following them. Thus, it makes absolutely no difference in the source file to write

```
Tweedledum and Tweedledee.
```

or

```
Tweedledum  and  Tweedledee.
```

ConTeXt considers these two sentences to be exactly the same. Therefore, if we want to introduce an additional blank space between the words, we need to use some ConTeXt commands that do this. Normally it will work with « \ », meaning a \ character followed by a blank space. But there are other procedures that will be looked at in chapter ?? regarding horizontal space.

The absorption of consecutive blank spaces allows us to write the source file by visually highlighting parts we would like to highlight, simply by increasing or decreasing the indentation used, with the peace of mind of knowing that it will not in any way affect the final document. Thus, in the following example

```
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 historia 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).
```

you can see how some lines are slightly indented to the right. These are lines that are part of the bits that will appear in italics. Having these indented helps (the author) to see where the italics end.

Some might think, what a mess! Do I have to bother with indenting lines? The truth is that this special indenting is done automatically by my text editor (GNU Emacs) when it is editing a ConT_EXt source file. It's that kind of small help that makes you choose to work with a certain text editor and not another one.

The rule that blank spaces are absorbed applies exclusively to consecutive blank spaces in the source file. Therefore, if an empty group (« {} »), is placed in the source file between two blank spaces, although the empty group will not produce anything in the final file, its presence will ensure that the two blanks are not consecutive. For example, if we write « Tweedledum {} and Tweedledee », we will get « **Tweedledum and Tweedledee** », where, if you look closely enough, you will see two consecutive spaces between the first two words.

The same happens with the reserved « ~ » character, although its effect is to generate a blank space even though it really isn't one: a blank space followed by a ~ will not absorb the latter, and a blank space after ~ will not be absorbed either.

4.2.2 Line breaks

In most text editors, when a line exceeds the maximum width, a line break is automatically inserted. We can also expressly insert a line break by pressing the « Enter » or « Return » key.

ConT_EXt applies the following rules to line breaks:

- a. A single line break is, to all intents and purposes, equal to a blank space. Therefore, if immediately before or after the line break there is any blank space or tab, these will be absorbed by the line break or the first blank space, and in the final document a simple blank space will be inserted.
- b. Two or more consecutive line breaks create a paragraph break. For this, two line breaks are considered to be consecutive if there is nothing but blank spaces or tabs between the first and second line break (because these are absorbed by the first line break); which, in short, means that one or more consecutive lines that are absolutely blank in the source file (without any character in them, or only with blank spaces or tabs) become a paragraph break.

Note that I said « two or more consecutive line breaks » and then « one or more blank consecutive lines », meaning that if we want to increase the separation between paragraphs, we do not do so simply by inserting another line break. For this we need to use the command that increases vertical space. If we only want one extra line of separation, we can use the `\blank` command. But there are other procedures for increasing vertical space. I refer to section ??.

On some occasions, when a line break becomes white space, we can end up with some undesirable and unexpected white space. Especially when we are writing macros, where it

is easy for a blank space to « sneak in » without us realising it. To avoid this we can use the reserved character « % » which, as we know, causes the line where it appears not to be processed, which implies that the break at the end of the line will also not be processed. So, for example, the command

```
\define[3]\Test{
  {\em #1}
  {\bf #2}
  {\sc #3}
}
```

that writes its first argument in italics, the second in bold and the third in small caps, would insert a blank space between each of these arguments, while

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

will not insert any blank space between them, since the reserved character % prevents line breaks from being processed and just become a blank space.

4.2.3 Rules/dashes

Dashes are a good example of the difference between a computer keyboard and printed text. On a normal keyboard, there is usually only one character for the dash (or rule, in typographic terms) which we call the hyphen or (« - »); but a printed text uses up to four different lengths for rules:

- Short rules (hyphens), like those used to separate syllables in hyphenation at the end of a line (-).
- Medium-sized rules (en dashes or en rules), slightly longer than the previous ones (–). They have a number of uses including, for some European languages (less so in English) the beginning of a line of dialogue, or to separate the lesser from the greater digits in a range in dates or pages; « pp. 12–33 ».
- Long rules (em dashes or em rules) (—), used as parentheses to include one sentence within another.
- Minus sign (−) to represent subtraction or a negative number.

Today, all the above and others besides are available in UTF-8 encoding. But since they can't all be generated by a single key on the keyboard, they are not so easy to produce in a source file. Fortunately, \TeX saw the need to include more rules/dashes in our final document than could be produced by the keyboard, and designed a simple procedure to do so. \ConTeXt has complemented this procedure by also adding commands that generate these various kinds of rules. We can use two approaches for generating the four kinds of rule: either the ordinary \ConTeXt way with a command, or directly from the keyboard. These procedures are shown in [table 4.2](#):

Type of rule	Appearance	Written directly	Command
Hyphen	-	-	<code>\hyphen</code>
En rule	—	--	<code>\endash</code>
Em rule	—	---	<code>\emdash</code>
Minus sign	-	\$-\$	<code>\minus</code>

Tableau 4.2 Rules/dashes in ConTeXt

The command names `\hyphen` and `\minus` are the ones normally used in English. While many in the printing industry call them «rules», T_EX's terms, namely `\endash` and `\emdash` are also common in typesetting terminology. The «*en*» and «*em*» are the names of units of measure used in typography. An «*en*» represents the width of an «n» while an «*em*» is the width of an «m» in the font being used.

4.3 Simple and multifile projects

In ConT_EXt we can use just one source file that includes absolutely all the contents of our final document as well as all the details relating to it, in which case we are talking about «simple projects», or, by contrast, we could use a number of source files which share the contents of our final document, and in this case we are talking about «multifile projects».

The scenarios where it is typical to work with more than one source file are as follows:

- If we are writing a document in which a number of authors have collaborated, each with their own part different from the others; for example, if we are writing a festschrift with contributions from different authors, or the number of a journal, etc.
- If we are writing a lengthy document where each part (chapter) has relative autonomy, so that the final arrangement of these allows for several possibilities and will be decided at the end. This occurs with relative frequency for many academic texts (manuals, introductions and the like) where the order of chapters may vary.
- If we are writing a number of related documents that share some style characteristics.
- If, put simply, the document we are working on is large, such that the computer slows down either when editing or compiling it; in this case, splitting

the material across several source files will considerably speed up the compilation for each.

- Also, if we have written a number of macros that we want to apply in some (or all) our documents, or if we have generated a template that controls or styles our documents and we want to apply these to them, etc.

4.4 Structure of the source file in simple projects

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:

- **The document preamble:** everything from the first line in the file up to the beginning of the « text » environment (`\starttext`).
- **The body of the document:** this is the contents of the « text » environment; or in other words, everything between `\starttext` and `\stoptext`.

```
% First line of the document

% Preamble area:
% Containing the global configuration
% commands for the document

\starttext % The body of the document begins here

...
... % Document contents
...

\stoptext % End of the document
```

Figure 4.1 file containing a simple project

In figure 4.1 we see a very simple source file. Absolutely everything before the command `\starttext` (which in the image is on line 5, counting only those with some text), constitutes the preamble; everything between `\starttext` and `\stoptext` constitutes the body of the document. Anything after `\stoptext` will be ignored.

The preamble is used to include commands that are to affect the document as a whole, the ones that determine its overall configuration. It is not essential to write any command in the preamble. If there is none, ConTeXt will adopt a default

configuration which is not very developed but could do for many documents. In a well-planned document, the preamble will contain all the commands affecting the document as a whole, like macros and customised commands to be used in the source file. In a typical preamble, this could include the following:

- An indication of the document's main language (See section ??).
- An indication of paper size (section ??) and page layout (section ??).
- Features of the documents main font (section ??).
- Customisation of the section commands to be used (section ??) and, if needs be, definition of new section commands (section ??).
- Layout of headers and footers (section ??).
- Settings for our own macros (section 3.7).
- Etc.

The preamble is intended for the overall configuration of the document; therefore nothing that is to do with the *contents* of the document, or processable text, should be there. In theory, any processable text included in the preamble will be ignored, although sometimes, if it is there, it will cause a compiling error.

The body of the document, framed between the `\starttext` and `\stoptext` commands includes the actual contents, meaning processable text, along with ConTeXt commands that should not affect the whole document.

4.5 Multifile management in T_EX style

In order to work with more than one source file, T_EX included the primitive called `\input`, which also works in ConTeXt, although the latter includes two specific commands that to some extent perfect the way `\input` functions.

4.5.1 The `\input` command

The `\input` command inserts the contents of the file it indicates. Its format is:

`\input FileName`

where *FileName* is the name of the file to insert. Note that it is not necessary for the file name to be enclosed between curly brackets, even though it will not throw an error if this is done. However, it should never be put between square brackets. If the file extension is « `.tex` », it can be omitted.

When ConTeXt is compiling a document and finds an `\input` command, it looks for the file indicated and continues compiling as if this file were part of the file that called it. When it finishes compiling it, it returns to the original file and continues from where it left off; the practical result is, therefore, that the contents of the file called by means of `\input` are inserted at the point where that is called.

The file called with `\input` must have a valid name in our operating system and no blank spaces within the name. ConTeXt will look for it in the working directory, and if it doesn't find it there, it will look for it in directories included in the variable of the `TEXROOT` environment. If the file is not ultimately found, it will produce a compilation error.

The most common use of the `\input` command is as follows: a file is written, let's call it « `principal.tex` », and this will be used as a container for calling, through the `\input` command, the various files that make up our project. This is shown in the following example:

```
% General configuration commands:

\input MyConfiguration

\starttext

\input PageTitle
\input Preface
\input Chap1
\input Chap2
\input Chap3

...

\stoptext
```

Note how, for the general configuration of the document, we have called the file « `MyConfiguration.tex` » which we assume contains the global commands we want to apply. Then, between the commands `\starttext` and `\stoptext` we call the several files that contain the contents of the various parts of our document. If, at a given moment, to speed up the compiling process, we want to leave out compiling some files, all we need to do is put a comment mark at the beginning of the line calling that or those files. For example, if we are writing the third chapter and we want to compile it simply to check that there are no errors in it, we don't need to compile the rest and therefore can write:

```
% General configuration commands:

\input MyConfiguration

\starttext

% \input PageTitle
% \input Preface
% \input Chap1
% \input Chap2

\input Chap3

...

\stoptext
```

and only Chapter 3 will be compiled. Note how, on the other hand, changing the order of chapters is as simple as changing the order of the lines calling them.

When we exclude a file in a multifile project from being compiled, we gain in processing speed, but as a result, all the references that the part being compiled makes to other parts not as yet compiled will no longer work. See section ??.

It is important to be clear that when we are working with `\input`, only the main file, the one that calls all the others, must include the `\starttext` and `\stoptext` commands, because if the other files include them, there will be an error. This, on the other hand, means that we cannot directly compile the different files that make up the project, but must necessarily compile them from the main file, which is the one that houses the basic structure of the document.

4.5.2 `\ReadFile` and `\readfile`

As we have just seen, if ConTeXt does not find the file called with `\input`, it will generate an error. For the situation where we want to insert a file only if it exists, but allowing for the possibility that it might not, ConTeXt offers a variation of the `\input` command. This is

`\ReadFile{FileName}`

This command is similar to `\input` in every respect, with the only exception that if the file to be inserted is not found, it will continue compiling without generating any kind of error. It also differs from `\input` in its syntax, since we know that with `\input` it is not necessary to put the file name of the file to be inserted between curly brackets. But with `\ReadFile` it is necessary. If we don't use curly brackets, ConTeXt will think that the name of the file to be sought is the same as the first character that follows the `\ReadFile` command, followed by the extension `.tex`. So, for example, if we write

`\ReadFile MyFile`

ConTeXt will understand that the file to be read is called «`M.tex`», since the character immediately after the command `\ReadFile` (excluding blank spaces that are, as we know, ignored at the end of a command name) is an «`M`». Since ConTeXt will not normally find a file called «`M.tex`», and `\ReadFile` does not generate an error if it doesn't find the file, ConTeXt will continue compiling after the «`M`» in «`MyFile`», and will insert the text «`yFile`».

A more refined version of `\ReadFile` is `\readfile` whose format is

`\readfile{FileName}{TextIfExists}{TextIfNotExists}`

The first argument is similar to `\ReadFile`: the name of a file enclosed between curly brackets. The second argument includes the text to be written if the file

exists, before inserting the contents of the file. The third argument includes the text to be written if the file in question is not found. This means that depending on whether or not the file entered as the first argument is found, the second argument (if the file exists) or the third (if the file does not exist) will be executed.

4.6 ConT_EXt projects as such

The third mechanism that ConT_EXt offers for multifile projects is more complex and complete: it starts by distinguishing between project files, product files, component files and environment files. To understand the relations and functioning of each of these types of file, I think it is best to explain them each individually:

4.6.1 *Environment files*

An environment file is a file that stores the macros and configurations of a specific style that is intended to be applied to several documents, whether they are completely independent documents or parts of a complex document. The environment file, therefore, can include everything we would normally write before `\starttext`; that is: the general configuration of the document.

I have retained the term « environment files » for these kinds of files, in order not to depart from the ConT_EXt official terminology, even though I believe that a better term would probably be « format files » or « global configuration files ».

Like all ConT_EXt source files, the environment files are text files, and assume that the extension will be « `.tex` », although if we want we can change it, perhaps to « `.env` ». Usually this is not done in ConT_EXt however. Most often the environment file is identified by starting or ending the name with «env». For example:« `MyMacros_env.tex` » or « `env_MyMacros.tex` ». The inside of such an environment file would look something like the following:

```
\startenvironment MyEnvironment

  \mainlanguage[en]

  \setupbodyfont
    [modern]

  \setupwhitespace
    [big]

  ...

\stopenvironment
```

Or in other words, definitions and configuration commands come within `\startenvironment` and `\stopenvironment`. Immediately following `\startenvironment` we write the name by which we want to identify the environment in

question, and then include all the commands we would like our environment to be made up of.

With regard to the name of the environment, according to my tests, the name we add immediately after `\startenvironment` is merely indicative, and if we were to give it no name, then nothing (bad) happens.

Environment files were intended to work with components and products (explained in the next section). This is why one or more environments can be called from a component or a product using the `\environment` command. But this command also works if it is used in the configuration area (preamble) of any ConTeXt source file, even if it is not a source file intended to be compiled in parts.

The `\environment` command can be called using either of the two following formats:

`\environment File`

`\environment[File]`

In either case, the effect of this command will be to load the contents of the file taken as an argument. If that file is not found, it will continue compiling in a normal way without generating any error. If the file extension is « `.tex` », it can be omitted.

4.6.2 Components and products

If we think of a book where each chapter is in a different source file, then we would say that the chapters are *components* and the book is the *product*. This means that the *component* is an autonomous part of a *product*, able to have its own style and to be compiled independently. Each component will have a different file, and, in addition, there will be a product file that brings all the components together.

A typical component file would be as follows

```
\environment MyEnvironment
\environment MyMacros

\startcomponent Chapter1

  \startchapter[title={Chapter 1}]

  ...

\stopcomponent
```

And a product file would look like the following:

```
\environment MyEnvironment
\environment MyMacros

\startproduct MyBook

  \component Chapter1
  \component Chapter2
  \component Chapter3

  ...

\stopproduct
```

Note that the actual contents of our document will be distributed among the various «component» files and the product file is limited to establishing the order of the components. On the other hand, the (individual) components and the products can be compiled directly. Compiling a product will generate a PDF file containing all the components of that product. If, on the other hand, one of the components is compiled individually, it will generate a pdf file containing only the compiled component.

Within a component file, and before the `\startcomponent` command, we can call one or more environment files with `\environmentEnvironmentName`. We can do the same in the product file before `\startproduct`. Several environment files can be loaded simultaneously. We can, for example, have our favourite collection of macros and the different styles we apply to our documents all in different files. Please note, however, that when we use two or more environments, these are loaded in the order in which they are called, so that if the same configuration command has been included in more than one environment, and it has different values, the values of the last loaded environment will apply. On the other hand, environment files are loaded only once, so in the previous examples in which the environment is called from the product file and from specific component files, if we compile the product, that is the time when the environments are loaded, and in the order indicated there; when an environment is called from any of the components, ConTeXt will check if that environment is already loaded, in which case it will do nothing.

The name of the component that is called from a product must be the name of the file that contains the component in question, although, if the extension of this file is «`.tex`», it can be omitted.

4.6.3 Projects as such

The distinction between products and components is sufficient in most cases. Just the same, ConTeXt has an even higher level where we can group a number of products: this is the *project*.

A typical project file would be more or less as follows

```
\startproject MyCollection

\environment MyEnvironment
\environment MyMacros

\product Book01
\product Book02
\product Book03

...

\stopproject
```

A scenario where we would need a project would be, for example, where we need to edit a collection of books, all with the same format specifications; or if we were editing a journal: the collection of books, or the journal as such, would be the project; each book or each journal issue would be a product; and each chapter of a book or each article in a journal issue would be a component.

Projects, on the other hand, are not intended to be compiled directly. Consider that by definition each product belonging to the project (each book in the collection, or each journal issue) should be compiled separately and generate its own PDF. Therefore the `\product` command included in it to indicate what products belong to the project, actually does nothing: it is simply a reminder for the author.

Clearly, some could ask why we have projects if they can't be compiled: the answer is that the project file links certain environments to the project. This is why, if we include the `\projectProjectName` command in a component or product file, ConTeXt will read the project file and automatically load the environments linked to it. This is why the `\environment` command in projects has to come after `\startproject`; however, in products and components, `\environment` has to come *before* `\startproduct` or `\startcomponent`.

Just like with the `\environment` and `\component` commands, the `\project` command allows us to indicate the project name either inside square brackets or not use square brackets at all. This means that `\project FileName` and `\Project[FileName]` are equivalent commands.

Summary of the different ways of loading an environment

It follows from the above that an environment can be loaded by any of the following procedures:

- a. By inserting the `\environment EnvironmentName` command before `\starttext` or `\startcomponent`. This will load the environment for compiling the file in question only.
- b. By inserting the `\environment EnvironmentName` command in a product file before `\startproduct`. This will load the environment when the product is compiled, but not if its components are compiled individually.
- c. By inserting the `\project` command in a product or environment: this will load all environments linked to the project (in the project file).

4.6.4 Common aspects of environments, components, products and projects

Names of environments, components, products and projects: We have already seen that, for all these elements, after the `\start` command that initiates a particular environment, component, product or project, its name is entered directly. This name, as a rule, must coincide with the name of the file containing the environment, component or product because, for example, when ConTeXt is compiling a product and, according to the product file must load an environment or component, we have no way of knowing which file that environment or component is unless the file has the same name as the element to be loaded.

Otherwise, according to my tests, the name written after `\startproduct` or `\startenvironment` in the product and environment files is merely indicative. If it is omitted, or does not match the name of the file, nothing bad happens. However, in the case of components, it is important that the name of the component matches the name of the file that contains it.

Structure of project directories: We know that by default ConTeXt looks for files in the working directory and in the path indicated by the `TEXROOT` variable. However, when we use the `\project`, `\product`, `\component` or `\environment` commands it is assumed that the project has a directory structure in which common elements are found in the parent directory, and the specific ones in some child directory. So, if the file indicated in the working directory is not found, it will be searched for in its parent directory, and if it is not found there either, in that directory's parents directory, and so on.

II

Global aspects of the document

III

Particular issues

Appendices

a

aide et ressources [29](#)

c

compilation [35](#)

composition [15](#)

d

define [61](#)

definestartstop [64](#)

l

langages de balisage

balises [17](#)

markup [17](#)

m

Mark II [14](#)

Mark IV [14](#)

moteurs T_EX [18](#)

p

préambule [38](#)

r

rédaction [15](#)

règles syntaxiques
commande [59](#)

s

setupwhitespace [70](#)